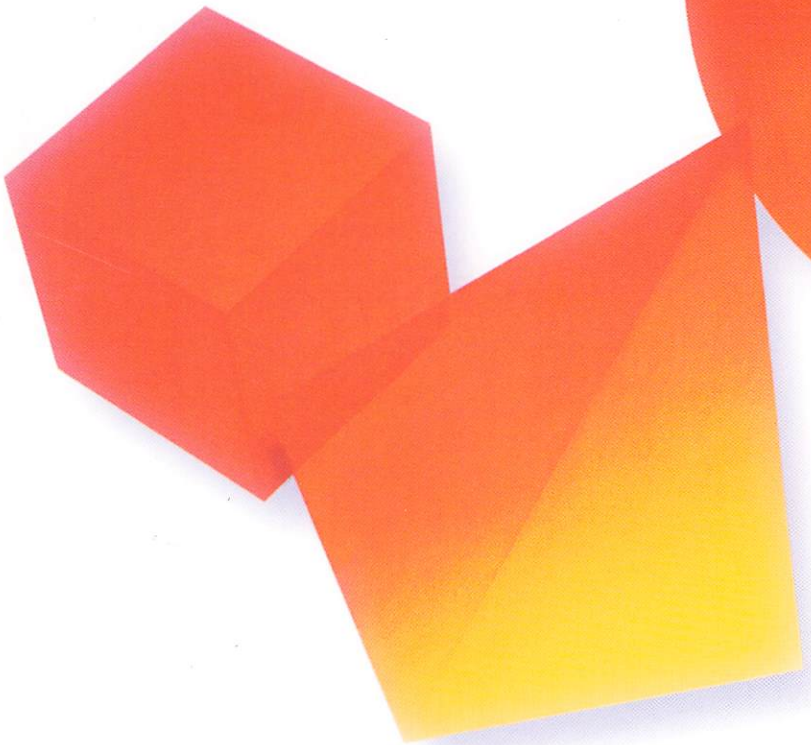
Quick Tour & Quick Reference

VERSION 5



Borland® C++

Quick Tour & Quick Reference

Introducing Borland C++ *2*

Looking inside Borland C++ *3*

Building Windows applications *4*

Managing projects *6*

Customizing your environment *8*

Automating application development *9*

Refining your application *10*

Designing a graphical user interface *11*

Using frameworks and libraries *12*

Developing database applications *19*

Debugging *20*

Working from the command line *22*

Programming with C and C++ languages *23*

Writing portable programs *25*

Learning more about Borland C++ *27*

Expanding the power of Borland C++ *29*

Quick Reference *31*

Index *50*

Introducing Borland C++

To install Borland C++ on your system, type:

D:\SETUP\BC5\SETUP.EXE

Welcome to Borland C++, the most complete C and C++ programming environment available today. With the Borland C++ visual development environment, creating complex applications is as simple as pointing and clicking on choices from a menu, or selecting items in a dialog box.

The Borland C++ compilers meet the ANSI C standard and provide the most conforming implementation of the latest draft ANSI/ISO C++ standard. At the same time, Borland's C++ provides innovative support for application development for Windows 95, Windows NT, Windows 3.x, Win32s, and DOS.

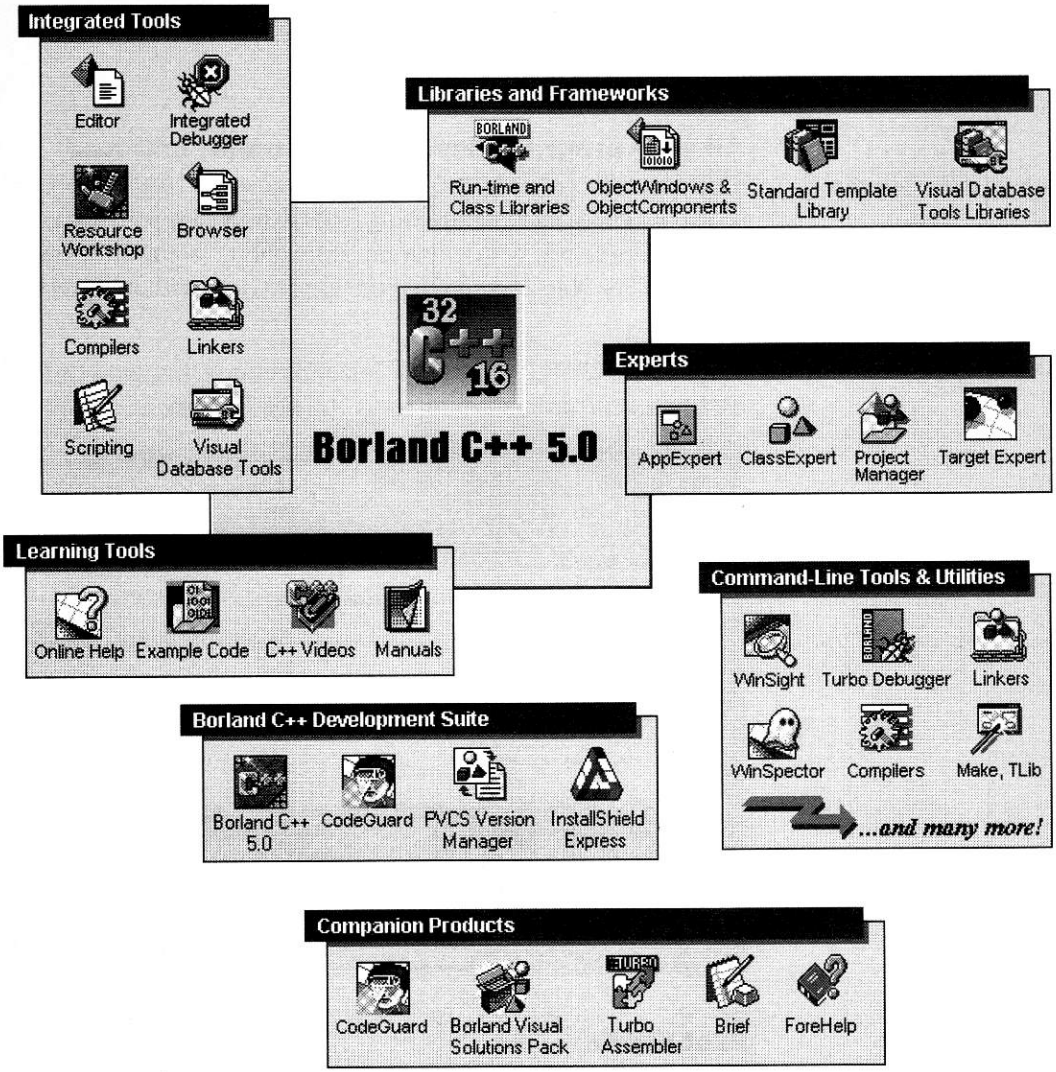
The integrated development environment (IDE) and tools reduce the time required to set up and code your application, allowing you to focus on getting results. Whether you are writing new code, maintaining existing applications, debugging, creating Windows resources, or migrating to another PC platform, Borland C++ technology gives you everything you need to work effectively. And with ObjectScripting, you can even customize and automate your development environment so it meets your individual needs.

With Borland C++, one powerful user interface gives you access to all the tools you need. Use the Project Manager to organize and manage your files. Use AppExpert to create the foundation of your Windows applications, and use ClassExpert to modify and refine these applications. Use Resource Workshop to visually design and create the pieces of your user interface.

Use the integrated debugger to step through and correct your code. Use Borland's Visual Database Tools to access extensive database support for your applications. These are just a few of the many tools Borland C++ provides to make application development easy and efficient.

The Borland C++ class libraries and the ObjectWindows and ObjectComponents frameworks give you access to a powerful collection of classes that solve many of your most difficult Windows programming tasks. ObjectComponents hides the complexity of OLE 2 programming and lets you write new OLE 2 applications or simply add this capability to your existing programs.

The complete documentation that comes with Borland C++ gives you access to the information you need quickly—through online Help and complete examples. The full documentation set is also available in print



This *Quick Tour* is an overview of Borland C++ and its complex feature set. Throughout this booklet, refer to the left margin for where to find details about the topics that interest you. You'll also find a handy quick reference at the back of this book.

Building Windows applications

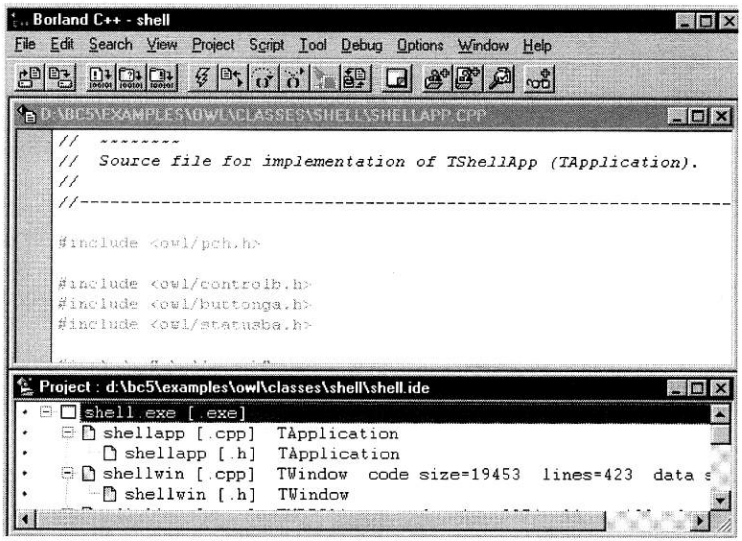
Borland C++ simplifies Windows application development by giving you one environment for all your programming tasks, Experts for quick project starts, example programs for coding ideas, the tools you need to create complete and efficient Windows applications, and much more.

All your program development under one roof

In the Help index, search for "IDE".

THE BORLAND C++ INTEGRATED DEVELOPMENT ENVIRONMENT (IDE) puts all your programming tools within easy reach—you can develop resources, write program code, debug, compile, browse through C++ objects, and reference expert Help without leaving the IDE.

You can use ObjectScripting to customize the IDE so you can call on any special programming tools you need to use.



Fast starts

THE BORLAND C++ EXPERTS and frameworks let you step right into your real programming challenges. Use the Experts so you don't lose time coding the basic functionality of your Windows applications.

Use the Frameworks so you don't spend time getting bogged down in the overhead inherent in Windows programming. These Borland C++ tools make it easier to make the transition to Windows programming.

Building Windows applications

The Borland C++ Experts

In the Help index, search for "TargetExpert", "AppExpert", and "ClassExpert".

The three Experts in Borland C++ give you an instant programming edge. Use TargetExpert to set up your project files. Use AppExpert to avoid mundane Windows programming chores by having it generate the source-code foundation for your projects. Use ClassExpert to customize the code generated by AppExpert.

The ObjectWindows framework

In Help, right-click and choose OWL.

Also, see "Using frameworks and libraries" in this booklet.

Use the ObjectWindows framework to simplify the complex nature of Windows programming. The numerous high-level objects in ObjectWindows let you focus on solving real problems instead of fighting the complexity often involved with Windows programming. Using the classes in ObjectWindows, you can manage your dialog boxes, scroll bars, bitmaps, and much more. You can even use ObjectWindows to plug VBX controls into your 16- and 32-bit Windows applications.

Productivity tools

In the Help index, search for "precompiled headers", "browser", and "editor".

WITH BORLAND C++, YOU CAN USE precompiled headers to compile your programs in a fraction of the time it normally takes. Then, use the Browser to view the hierarchy of your classes. From the Browser, a click of the mouse moves to any file in your project or class.

The Borland C++ editor is fully configurable; you can customize the BRIEF-like editor by changing the key mappings, the syntax highlighting, and the editor SpeedBar. The editor also supports multiple files, and you can view them all simultaneously by splitting the Edit window into multiple panes.

Visual interface design

See "Designing a graphical user interface" in this booklet.

In the Help index, search for "resources".

RESOURCE WORKSHOP IS AN INTEGRATED VISUAL TOOL that helps you design your user interface by displaying the layout of your UI components as you create them. This helps you make the best use of your time: you don't need to go back and forth between a resource compiler and your application to view your UI.

Using even more tools

In the Help index, search for "tools".

See "Customizing your environment" in this booklet.

YOU CAN CONFIGURE THE BORLAND C++ IDE to include custom tools, translators, and viewers on the Tools menu. Tools are standalone programs like Turbo Debugger; translators are programs that generate one file type from another; and viewers are programs that let you examine files of a particular type.

For more information...

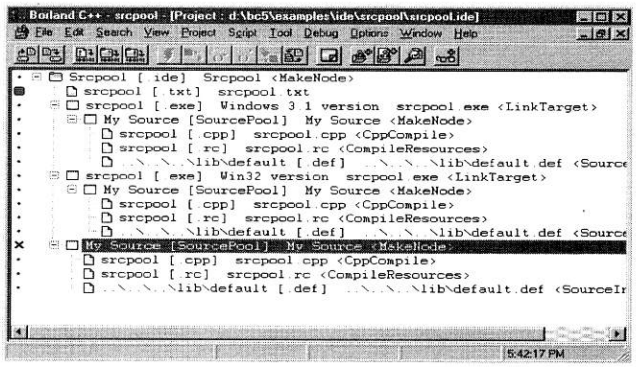
Managing projects

See "Managing Projects" in the *User's Guide*.

The Borland C++ Project Manager makes it easy for you to manage your C and C++ projects. Whether your projects are large or small, and whether they target Windows 95, Windows NT, Windows 3.x, Win32s, or DOS, the Project Manager handles them all.

Seeing the whole project

THE PROJECT MANAGER DISPLAYS all your project files in a hierarchy to give you a visual overview of your project organization.



In the Help index, search for "project hierarchy".

You can customize the Project Manager to display only the files you want to see—with the flip of a switch you can view or hide the source-file dependencies (such as header files), the libraries and startup modules, or the object files contained in your project.

Accessing project source files

In the Help index, search for "editor".

SIMPLY DOUBLE-CLICK A SOURCE-FILE NODE in the Project window to bring that file into an active Edit window. The integrated editor lets you view or modify the contents of any program header file or source file. You can also use the editor to create and modify resource scripts, ObjectScripting files, and other ASCII text files.

Creating multipurpose projects

See SRCPOOL.IDE in EXAMPLES\IDE\SRCPOOL.

In the Help index, search for "Source Pools".

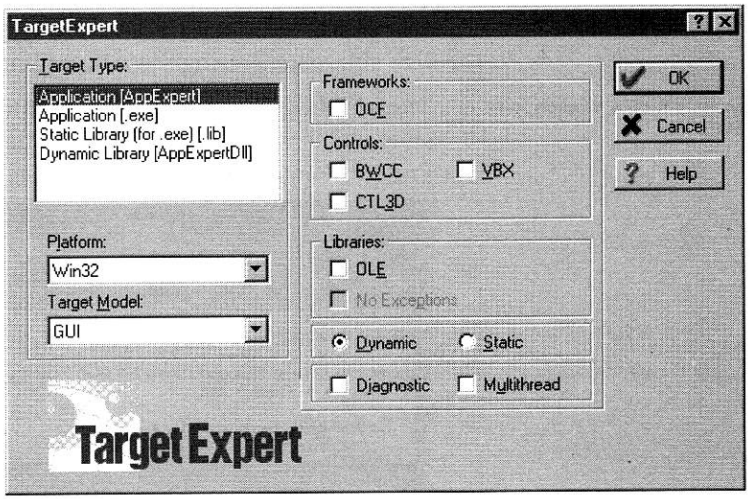
THE PROJECT MANAGER LETS YOU CREATE multiple 16- and 32-bit applications and DLLs from a single project. This gives you the ability to generate both a 16- and a 32-bit application with a single command. For example, you can build two versions of a single application: one that contains debug information, and another that doesn't.

The Project Manager also supports Source Pools to help you organize the files that different targets share. Grouping and sharing core source files in multitargeted projects makes it easy to manage these files as your projects become increasingly large and complex.

Linking the correct libraries

In the Help index, search for "TargetExpert".

TARGETEXPERT SETS UP THE CORRECT LIBRARIES to link, whether you're building a Windows import library, a DOS overlay module, or a 32-bit Windows application. When you use TargetExpert, you never have to remember the names of library files.



With TargetExpert, you select the target type, platform, target memory model, controls, and types of libraries to use. Then, let TargetExpert configure your project with the appropriate libraries and startup modules.

Setting project options

In the Help index, search for "Project options" and "project tree".

THE BORLAND C++ IDE HELPS YOU set and modify project options with the Project Options dialog box. Once set, you can quickly view and modify your option settings with the Options Hierarchy.

The Project Manager lets you set project options for an entire branch in the project hierarchy, and it lets you override the options for any single file in the project. This flexibility gives you the ability to create customized project settings for the most demanding compile and link situations you will encounter.

In the Help index, search for "Style sheets".

Also, see STYLESHT.IDE in EXAMPLES\IDE\STYLESHT.

To simplify the task of setting options, the Project Manager provides Style Sheets so you can group option settings into distinct units. By creating several Style Sheets, you can apply a set of options to any node in your project.

Customizing your environment

Borland C++ provides a 32-bit hosted Integrated Development Environment (IDE) that can be custom tailored to fit your individual needs using ObjectScripting.

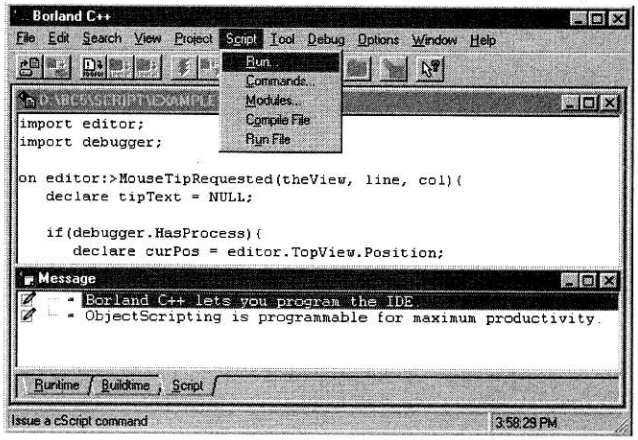
Run scripts to add features to the IDE

Refer to the *ObjectScripting Programmer's Guide.*

USING OBJECTSCRIPTING, you can customize the Borland C++ IDE using built-in classes and a C++-like scripting language called cScript. cScript offers fast execution speeds and extensive access to IDE subsystems using the IDE scripting classes.

In the Help index, search for "script".

The scripting classes provide access to the IDE's important subsystems, such as the editor, the debugger, keyboard, and the Project Manager, and you can change them to suit your needs.



Automate your work

See ..\SCRIPT\EXAMPLES for lots of sample scripts.

WITH OBJECTSCRIPTING, you can automate tedious manual tasks, integrate tools and utilities into the IDE, and even add new custom features. Extensive online Help and the *ObjectScripting Programmer's Guide* provide the information and examples you need to get started writing scripts to simplify your work.

Let your imagination run wild

See ..\SCRIPT\EXAMPLES\Tt.spp for a script that does this.

THERE'S LOTS YOU CAN DO. With ObjectScripting, you can add new features like a context-sensitive debugging watch window that shows the current value whenever the pointer is placed over a variable.

You could also create a feature that recognizes function names as you type them and automatically inserts the rest of the function prototype and the appropriate include file. This would save you from having to look up the prototypes and add new include files.

For more information...

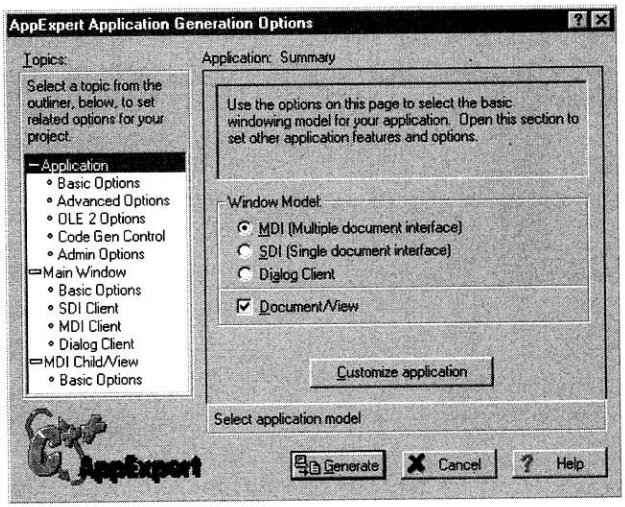
Automating application development

Borland C++ provides the Experts you need to get a jump on your Windows programming tasks. Use AppExpert to generate the source-code foundation of your application, then use ClassExpert to add the functionality that produces a custom application.

Generating Windows applications

In the Help index, search for "AppExpert".

LET APPEXPERT PROVIDE THE SOURCE-CODE FOUNDATION for everything you need in a Windows application. With AppExpert, a click of the mouse brings you the source code for online Help, toolbars, menus, status lines, and many other Windows application features.



In the Help index, search for "AppExpert", then "Building Applications with AppExpert".

Instead of coding the foundation of your Windows program yourself, use AppExpert to select the application type, the window features, and the main features of your program. Then click the Generate button and let AppExpert create a complete and working object-oriented Windows application. It saves time and reduces the chance of errors by providing a source-code foundation that you can work with.

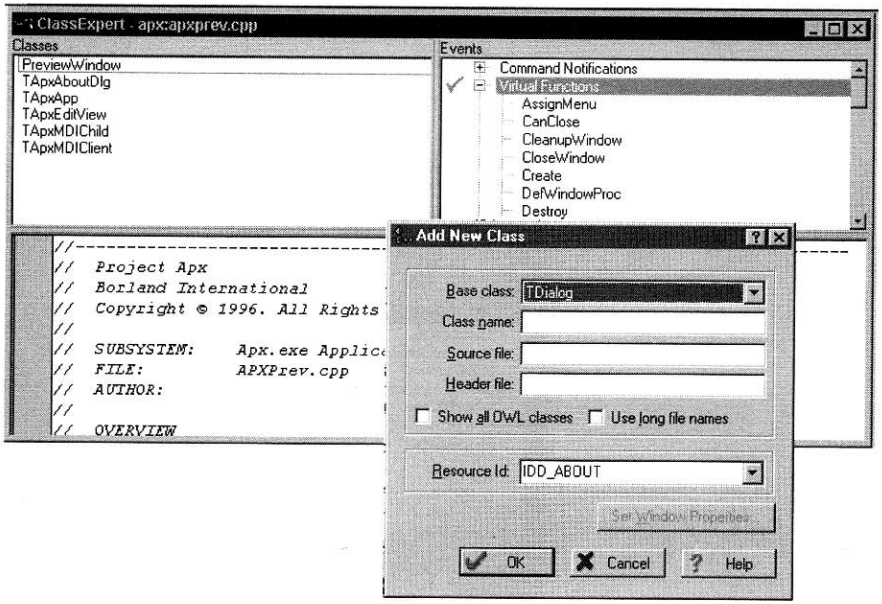
Programming visually

APPEXPERT GETS YOU UP AND RUNNING by generating all the foundation code for your MDI, SDI, and Dialog Client applications. You can create Doc/View applications and applications that support both the container and server models of OLE 2. All this is done by pointing and clicking—you don't need to type a single line of code to create a working ObjectWindows application.

Fine-tuning your AppExpert applications

See the *User's Guide*. In the Help index, search for "ClassExpert".

USE CLASSEXPERT TO CREATE NEW CLASSES, event handlers, and member functions. You can even use ClassExpert to locate existing classes in your code and to customize the generic functionality of the classes generated by AppExpert.



ClassExpert is the tool you use to modify the source code generated by AppExpert. Click on the event whose functionality you want to enhance, and ClassExpert opens a full-featured editor with the insertion point indicating where to place new code. In fact, because of its BRIEF-like editor and code navigation capabilities, ClassExpert is an ideal environment for developing ObjectWindows programs.

You can use ClassExpert to represent and automate classes in AppExpert projects. You can also tap into existing programs using Rescan to incorporate the code into your new AppExpert applications.

Adding resources to your applications

See the *Resource Workshop User's Guide*.

CLASSEXPERT WORKS SEAMLESSLY WITH RESOURCE WORKSHOP to associate resources with your ClassExpert classes. This lets you visually design the user interface for your AppExpert projects.

For more information...

Designing a graphical user interface

When you're ready to design the user interface for your application, use Resource Workshop from right within the IDE. Its graphical environment makes it easy to design user interface components just as they will appear in your application. You can also use Resource Workshop to develop database applications.

You can easily create a new Resource Project from the File menu. While working in the project, you can easily add new resources or modify existing ones.

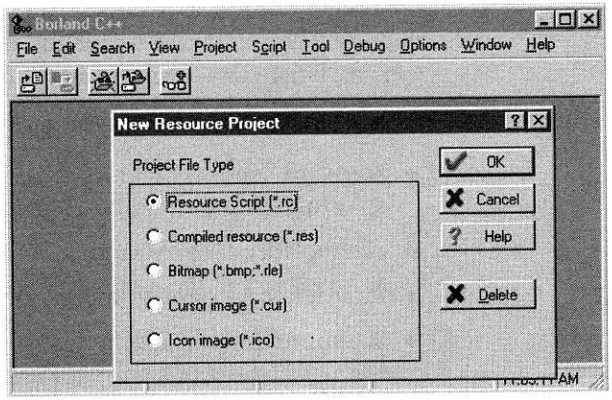
Types of user interface components

See the *Resource Workshop User's Guide*.

In the Help index, search for "resources".

RESOURCE WORKSHOP PROVIDES THE TOOLS you need to create and edit a variety of user interface components, including the following:

- **Dialog boxes.** Create dialog boxes that pop up when a menu command is chosen or when an event occurs in an application. Or, create a dialog box that serves as the main window of an application.
- **Menus.** Create standard menus that appear on the menu bar of an application, or floating menus that appear anywhere on the screen.
- **Graphics.** Create bitmaps (.BMP files), icons (.ICO files), and cursors (.CUR files).



Linking user interface components to code

In the Help index, search for "resource script".

WHEN YOU'RE READY TO LINK user interface components to code, create a resource script (.RC) file using Resource Workshop. Use the IDE or command-line tools to compile the resource script into a binary resource (.RES) file, then link the .RES file to an application. Using ClassExpert, you can automatically generate ObjectWindows classes that correspond to user interface components.

Using frameworks and libraries

See the *ObjectWindows Programmer's Reference.*

Frameworks and run-time libraries support a wide variety of computational tasks. Each framework is an object-oriented collection of C++ classes and routines designed to solve a particular programming need. Borland C++ frameworks provide a solid foundation upon which to build your application. From within your application, you can use the classes, routines, and functions available in the libraries to perform a wide variety of tasks.

See the MFC.TXT file in BC5\DOC.

You can also build Microsoft® Foundation Class (MFC) programs with Borland tools. If you have the MFC library source code, you can rebuild the MFC library with Borland C++.

Using the Standard C++ Library

See the *Standard C++ Library Programmer's Reference.*

BORLAND C++ INCLUDES Rogue Wave®'s implementation of the Standard C++ Library. This large library supports the latest C++ standardization efforts and provides a comprehensive collection of classes and functions. The Standard C++ Library is rapidly becoming the standard set of classes and libraries delivered with all ANSI-conforming C++ compilers.

Use Standard C++ Library components directly when you need flexibility or highly efficient code. For example, if you need an efficient data structure or algorithm to solve a compact problem, such as data stream manipulation in drivers (the kind of problem that often resolves to a single class), look to the Standard C++ Library. The Standard C++ Library excels in the creation of reusable classes, where low-level constructs are needed.

The current version of the Standard C++ Library includes:

- Standard Template Library (STL), which consists of data structures, algorithms, and iterators.
- A templatized *string* class.
- A templatized class for representing *complex* numbers.
- A uniform framework for describing the execution environment, through the use of a template class named *numeric_limits* and specializations for each fundamental data type.

Borland Component Libraries (BCL)

See the *ObjectWindows Programmer's Reference.*

THE BORLAND COMPONENT LIBRARIES contain three parts:

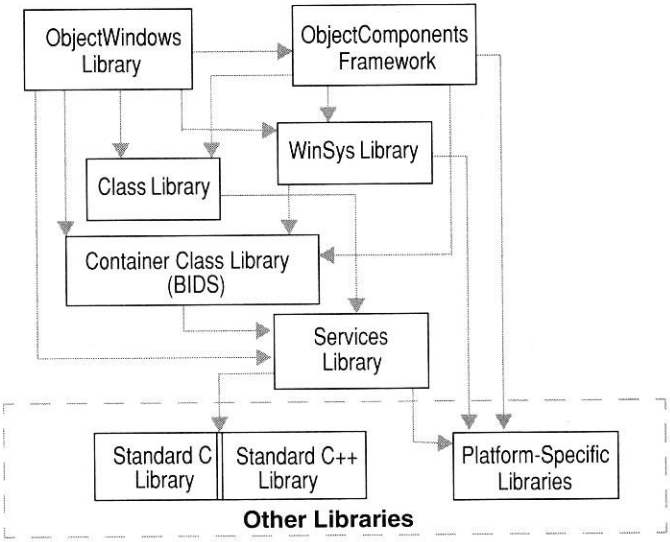
- ObjectWindows Library (owlxx.lib)
- ObjectComponents Framework (ocf.lib)
- WinSys, Class, and Services Libraries (bidsxx.lib)

Using frameworks and libraries

All the parts of the Borland Component Libraries provide lots of flexible components that make your programming easier. The ObjectWindows library contains the high-level C++ classes for Windows programming. The ObjectComponents library contains classes for creating OLE and automation components. Low-level windowing system classes are contained in the WinSys library.

The following figure shows the relationships between the various libraries.

Borland Component Libraries



The Class Library contains general application support classes, the object streaming system, and the Borland templatized container classes. The Services library is a thin layer of headers that presents a common system header encapsulation for the rest of BCL.

Professional applications with ObjectWindows

In Help, right-click OWL and select Task Help in the Contents.

OBJECTWINDOWS IS AN EASY-TO-USE, yet powerful and flexible, object-oriented C++ framework that encapsulates complex windowing functionality. ObjectWindows helps you develop applications that run faster, write code that's easier to debug, and create programs that implement linking and embedding technology. Use ObjectWindows to develop both 32- and 16-bit applications.

ObjectWindows supports standards like Windows 95-based common controls, and extends their power with features such as a splash screen, dockable toolbar, animation, and splitter window.

Using frameworks and libraries

Want to see an ObjectWindows application at work? Take a look at the Borland C++ IDE. It was written using ObjectWindows. Lots of examples are also located in the BC5\EXAMPLES\OWL directory.

Internet programming

In the Help index, search for "Windows Sockets".

ObjectWindows provides Windows Sockets classes. The Windows Sockets classes enable an application to communicate over the Internet, using the TCP/IP protocol. Both stream (TCP) and datagram (UDP) sockets are supported.

Adding event handlers to your application

In the Help index, search for "event handlers".

To add event handlers and messages to your application, use ObjectWindows response tables, which map Windows messages to the correct C++ member functions. You can also gain the functionality of event handlers derived from more than one class.

Handling exceptions

In the Help index, search for "exception handling".

To trap errors that might unexpectedly occur in your program, use ObjectWindows exception-handling objects. ObjectWindows has built-in exception-handling classes that treat exceptions in a consistent, predictable manner.

Handling documents and views

In the Help index, search for "Doc/View".

You can easily create applications based on the abstract Doc/View model of data handling and display functions. ObjectWindows separates the data from its presentation so you can create multiple views of the same data. When the data is changed in any one of the views, all of the views are updated automatically.

Doc/View applications are flexible. You can easily transform a Doc/View application into an OLE container or server.

Diagnostics

See ..\EXAMPLES\OWL\TOOLS\DIAGXPRT.

For greater programming safety and increased debugging support, ObjectWindows includes several internal diagnostic classes. You can use these diagnostic classes to catch usage errors or incorrect computations. Built-in diagnostic messages tell you if you have application, window, Doc/View, or GDI-related errors.

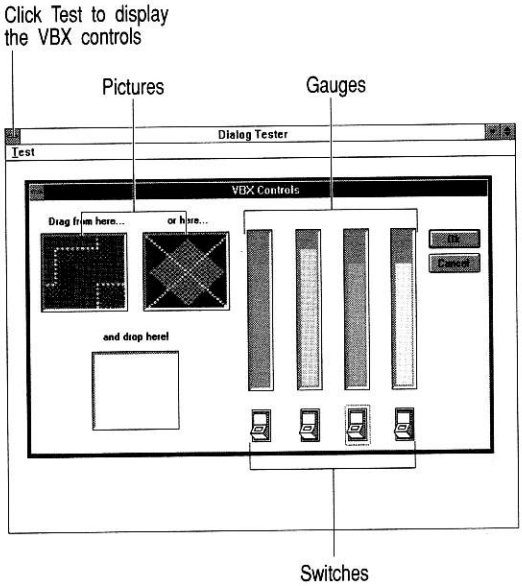
Configuring controls and windows

In the Help index, search for "VBX controls".

You can incorporate VBX controls from a wide variety of vendors into your 16- and 32-bit applications. ObjectWindows handles VBX controls just like standard ObjectWindows controls. It also provides drag-and-drop support for VBX controls.

Using frameworks and libraries

The rich set of toolbar, gadget, and dialog classes in ObjectWindows reduces the amount of code you have to write to create sophisticated user interfaces. Toolbar and gadget classes let you customize your user-interface objects. You can easily handle dialog boxes and their child controls in an object-oriented manner.



In the Help index, search for "layout windows".

Just as you can place toolbars and palettes with respect to their parent windows, you can also position other windows in relation to parent windows. Specialized window classes, such as layout windows, let you design constraint-driven windows whose size and position vary as the controlling window's parameters change.

Docking classes can be used in applications to position toolbars and status bars via drag-and-drop operations. Also, shell classes allow a program to manipulate objects in the Windows shell.

Enhancing your application

In the Help index, search for "Clipboard".

By using ObjectWindows Clipboard classes, you can browse the contents of the Clipboard and copy data to and from the Clipboard. You can use ObjectWindows specialized printer classes to add printer support with print previewing to an application.

OLE 2 encapsulation

In the Help index, search for "OLE".

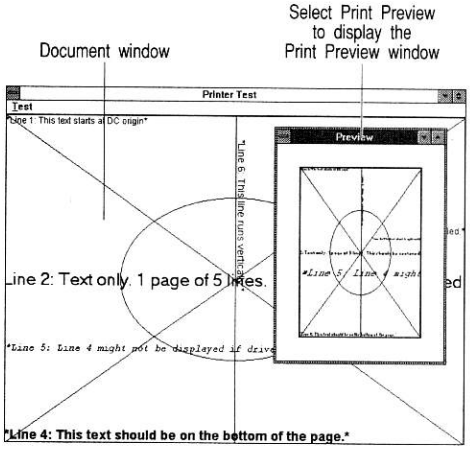
You can use the standard ObjectWindows OLE classes by themselves so your application can take advantage of OLE 2 features, such as in-place editing, embedding and linking, menu merging, and OCX controls.

Linking and embedding with ObjectComponents

In Help, right-click OCF.

USE THE OBJECTCOMPONENTS FRAMEWORK to write portable OLE 2-enabled programs. ObjectComponents is a set of classes for creating OLE 2 applications in C++.

You can even use ObjectComponents to add OLE 2 support to an existing C++ application.



With ObjectComponents, you can create applications containing spreadsheets from Quattro Pro, text from Microsoft® Word or WordPerfect, or objects exported from OLE servers, such as Paradox® or Visio. ObjectComponents also makes it easy to automate your application, so a server application can accept commands from other applications, and a controller application can control the behavior of other applications.

ObjectComponents and OLE 2 compatibility

In Help, right-click OCF and select Tasks.

Because ObjectComponents defines standard user interface features, you can easily add OLE functionality to existing C++ applications. In fact, ObjectComponents applications are compatible with other OLE applications, including OLE 1 applications, even if they weren't built using Borland tools.

Linking and embedding objects

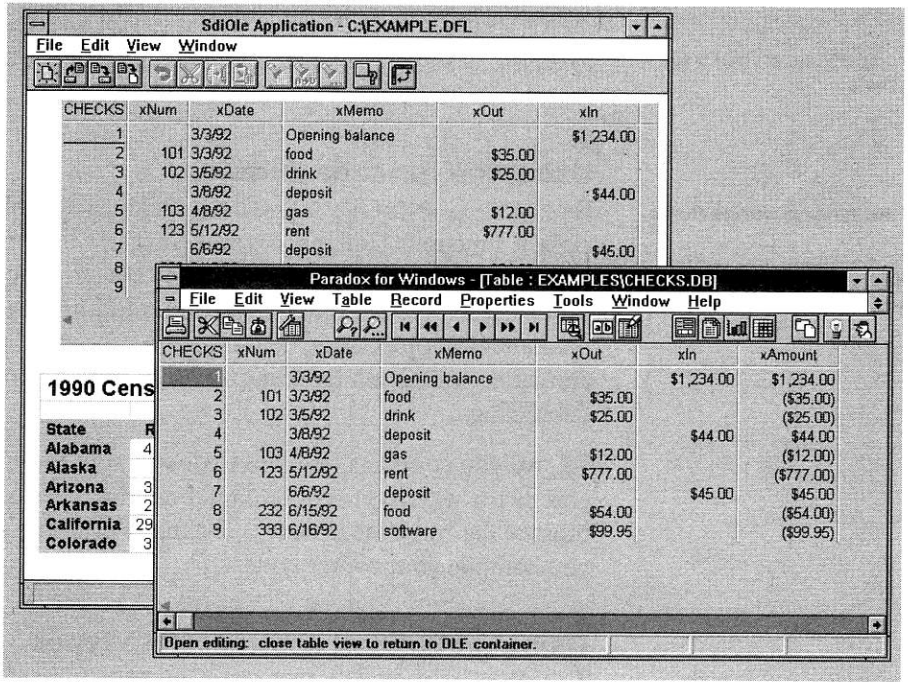
In Help, right-click OCF.

Using ObjectComponents classes, you can link and embed data from one application into another. To simplify this task, ObjectComponents provides default handlers that cut and paste objects onto the Clipboard. You can easily edit objects in place and merge server and container menus into one.

Create OLE 2 applications using the Experts

See "Automating application development" in this booklet.

To further simplify the process of creating OLE 2-enabled applications, use the Borland AppExpert and ClassExpert, which understand the new ObjectComponents classes.



Automation

See the *ObjectWindows Programmer's Reference.*

Add automation to your program so it can talk to other programs. ObjectComponents automates the C++ classes in your application without making structural changes to the classes themselves. Once your program is automated, other programs can talk to it through OLE commands.

In the Help index, search for "OCX container".

Your applications can link and embed OCX controls. ObjectComponents provides several new classes for turning an application into an OCX container.

Registration and translation

In the Help index, search for "registration macros".

Because of the flexible, yet powerful, ObjectComponents framework, many complex OLE transactions are transparent to you, the developer. For example, ObjectComponents registers an application with the OLE registry and run-time system. Using the localized information you provide, ObjectComponents looks up commands and descriptions in whatever language you request.

Examples provide more tools

See ..\EXAMPLES\OWL\TOOLS for tools for debugging ObjectWindows programs.

See ..\SOURCE\OCTOOLS for OLE tools.

YOU CAN FIND MORE TOOLS to help with ObjectWindows and ObjectComponents development. Tracing tools for debugging and viewing the internals of ObjectWindows programs are located in ..\EXAMPLES\OWL\TOOLS. Additional OLE utilities such as those to register a server, run a DLL server, and generate automation macros are located in ..\SOURCE\OCTOOLS.

Using the run-time libraries

See the *Class Libraries Guide*.

In the Help index, search for "run-time libraries".

BORLAND C AND C++ RUN-TIME LIBRARIES are collections of classes, functions, and macros that you can call from within your C and C++ programs to perform a wide variety of processing tasks. These tasks include single thread and multithread processing, mathematical processing, floating-point processing and coprocessor emulation, exception handling, console and GUI initialization, and debugging.

To give you complete freedom to design applications that work exactly the way you need, Borland provides the source code used in most of the run-time libraries. The many available makefiles help you compile your source code.

Using predefined abstract data types

See the *Library Reference*, Chapter 3.

In the Help index, search for "container classes".

THE BORLAND CONTAINER CLASS LIBRARY is a collection of C++ template classes that implements commonly used data structures, such as arrays, lists, iterators, stacks, and queues.

The container classes allow flexibility in how you implement particular data structures. For example, by just changing a type definition, you can switch your stack implementation from a vector-based stack to a list-based stack. All member functions of the stack template class remain the same regardless of the implementation.

Although the container class library is still fully supported, you should consider using the Standard C++ Library for future application development. This will ensure program portability in the future.

For more information...

Developing database applications

Look for sample applications in ..\VDBT\EXAMPLES.

Borland C++ comes with Visual Database Tools. These tools enable C++ programmers to create robust 32-bit or 16-bit database applications quickly and easily.

Access to other databases

YOUR DATABASE APPLICATIONS CAN INTERACT directly with data created with desktop databases such as Paradox and Visual dBASE. Your application can also interact with ODBC data sources. Using Borland's SQL Links, your Windows applications can connect to remote SQL database servers such as Oracle, Sybase, Microsoft SQL Server, and InterBase.®

In the Help index, search for "Visual Database Tools".

Your database application uses Visual Database Tools to interact with the 32-bit or 16-bit Borland Database Engine (BDE). Visual Database Tools is built on top of the BDE, so you can create database applications without needing to know about the BDE.

Visual database programming tools

YOU USE THE VISUAL DATABASE TOOLS components to design your database forms within the Borland C++ IDE. You add database support to your applications by dragging a component onto a form in the resource editor and setting its properties.

Data-access components, consisting of tables, queries, data sources, stored procedures, sessions, fields, and so on, encapsulate the Borland Database Engine. You can access the data through these components with a simple point and click.

A rich set of data-aware controls, such as list boxes, edit controls, data grids, memo controls, check boxes, and radio groups, provide user-interface components that users use to view and modify data in a database. You, the programmer, can view the data live within the data-aware controls from within the C++ IDE. This way you can test your application as you're developing it.

OLE support

VISUAL DATABASE TOOLS ALSO USE the Component Object Model, the underlying architecture of OLE. Any OLE automation controller—even a spreadsheet, database, or word processor—can access data.

Debugging

Borland C++ provides a suite of debuggers and debugging tools to help you locate and eliminate errors from your program code.

Debuggers

In the Help index, search for "debugging".

WHETHER YOU'VE STUMBLED on a logic error or a run-time error, you can use the Borland debuggers to help you quickly locate the source of the problem.

Borland C++ comes complete with a debugger that's fully integrated into the IDE and a set of debuggers that can be run from outside the IDE. While you can use the integrated debugger to uncover most of the errors in your program, you can use the standalone Turbo Debuggers for special circumstances.

The integrated debugger

See "Debugger" in the *User's Guide*.

The integrated debugger lets you debug 32-bit Windows programs without leaving the IDE. You can execute your program line by line, inspect data elements and structures, and modify variables to see how different values affect program behavior.

Completely revised, the integrated debugger now offers a CPU window to give you an assembly-level view of your code. The new integrated debugger also offers a rich set of breakpoint options; you can now set breakpoints on the following program elements and events:

- Source code lines
- Program addresses
- Changing data items
- Program threads
- C++ exceptions
- Operating system exceptions
- Module load operations

Standalone Turbo Debugger

Use the Help menu in the Turbo Debugger.

Turbo Debugger is a set of debugging tools that supplements the integrated debugger. With Turbo Debugger, you get three different debuggers for debugging 16- and 32-bit Windows programs and DOS programs. You can now run TDW.EXE under Windows 95 and Windows NT to debug your 16-bit Windows programs.

Like the integrated debugger, you can use Turbo Debugger to set breakpoints and watches, step through code, examine the values of expressions, and modify expression values as the program runs. In addition, you can also use Turbo Debugger for:

Debugging

- Just-In-Time debugging (with TD32)
- Dual-monitor debugging
- Remote-machine debugging

WinSpector

See the *WinSpector User's Guide.*

DOES YOUR PROGRAM TERMINATE with a General Protection (GP) fault? WinSpector and its utilities are debugging tools that help you perform postmortem examinations of Windows programs that end with a GP fault. It creates a log file to help uncover the error.

In the Help index, search for "WinSpector".

WinSpector includes pre-processing utilities to enhance the reports it creates: BUILDSYM, TMAPSYM, and EXEMAP. These utilities, make public functions, variable names, and functions in the entry table of the executable available to WinSpector.

WinSpector also has a post-processing utility, DFA, that utilizes Turbo Debugger symbolic information to further enhance the readability of the GP fault reports it creates.

WinSight

See the *WinSight User's Guide.*

In the Help index, search for "WinSight".

WINSIGHT GATHERS INFORMATION about window classes, application windows, and window messages. With WinSight, you can study how an application creates its classes and windows, and monitor how different windows send and receive messages.

WinSight is a passive observer—it displays information about messages, but does not prevent messages from getting posted to the appropriate applications. Use WinSight first to find the windows you want to monitor, then to trace the messages that get sent.

CodeGuard

Ask for Borland CodeGuard or purchase the Borland C++ Development Suite.
See "Expanding the power of Borland C++" later in this booklet.

ADDING CODEGUARD to Borland C++ helps you to detect run-time errors such as memory overruns, invalid data references, uninitialized data access, and memory leaks. This powerful debugging tool is an add-on to Borland C++.

Turbo Profiler

See the *Turbo Profiler User's Guide.*

In the Help index, search for "Turbo Profiler".

USING THE STATISTICAL REPORTS generated by Turbo Profiler, you can discover the elusive bottlenecks that sap the speed of your 16-bit programs. You can then fine-tune your programs and improve their performance in an informed, controlled manner.

You can also use Turbo Profiler to perform coverage analysis, which detects if and when the routines in your program get executed.

Working from the command line

When you work in the IDE, much of the guesswork of selecting the right compiler and options is done for you. But if you prefer to work from the command line, Borland C++ provides fast, direct-access tools for compiling, linking, debugging, and managing projects.

Using the compilers and linkers

See the online file README.TXT for the latest details.

In the Help index, search for "command-line compiler".

BORLAND C++ PROVIDES YOU WITH 16- and 32-bit command-line compilers and linkers that let you develop your DOS and Windows programs from the command line. With the 32-bit command-line tools, you can also build 32-bit GUI applications and console-mode applications for Win32s and Windows NT.

The enhanced 16-bit linker provides post-link optimization that removes redundant information in executable files. File size is smaller and application startup time is reduced.

Using resource tools

In the Help index, search for "BRCC", "RLINK", "BRC", or "resource tools".

RESOURCE COMPILERS, LINKERS, AND SHELLS are also available from the command line:

- **Resource Compilers** compile resource script files and produce binary .RES files.
- **Resource Linkers** bind resources, in .RES file form, to an executable file and mark the resulting file as a Windows executable.
- **Resource Shells** are shells that let you start both BRCC (or BRCC32) and RLINK (or RLINK32) in a single step.

Managing your project with MAKE

In the Help index, search for "MAKE options".

TO FREE YOURSELF from having to track how your project gets built, you can write project information into a file. By writing into this file the conditions under which each part of your project is built, you can let your system and the MAKE utility decide when to update your project.

You define the rules MAKE uses to determine which files to build and how to build them. Your instructions can be as complex or as simple as needed to accomplish your programming tasks.

Programming with C and C++ languages

More than any other compiler, Borland C++ gives you easy access to the programming power of the evolving C++ language.

When you're developing cutting-edge applications, you need a programming language that's as easy as possible to use. To design high-quality applications, your programming language must be simultaneously hardware independent yet capable of exploiting all machine resources.

Borland C++ gives you exactly the kind of language support you need. The world leader in C and C++ implementation gives you industry standards, forward-looking technology, and compatibility.

ANSI C

See the *Programmer's Guide*. In the Help index, search for "ANSI".

BORLAND C++ COMPILERS ARE FULLY compliant with the ANSI C X3J11/88-159 Standard. Only Borland C++ is certified by BSI/ISO for C compliance.

Any C program that complies with the ANSI Standard can be compiled with Borland C++ compilers. Also, a program compiled under Borland C++ with language extensions disabled is highly portable to other ANSI C compilers.

C++

In the Help index, search for "C++ Options".

See "Using frameworks and libraries" in this booklet.

BORLAND CLOSELY TRACKS the evolving ANSI/ISO committee definition of the C++ language. Just a few C++ features that Borland C++ implements are:

- **Standard C++ Library** provides many classes and functions.
- **Namespace support** solves the problem of identifier name clashes in large applications.
- **Templates** to develop a family of related functions or classes.
- **Exception handling** provides a standard method to handle run-time errors. Only Borland C++ provides a way to mix structured exceptions developed in C with C++ code.
- **New-style type-safe casting** makes safe type conversions even when you're working with polymorphous classes.
- **Run-time type identification (RTTI)** determines the type of a data object at run time: virtual, reference, or pointer.
- **Three distinct character types** overload functions based on a distinction between **char**, **signed char**, and **unsigned char.**
- **New keywords and macros**, such as **bool**, **mutable**, **explicit**, **__declspec**, and **typename**, provide greater flexibility.

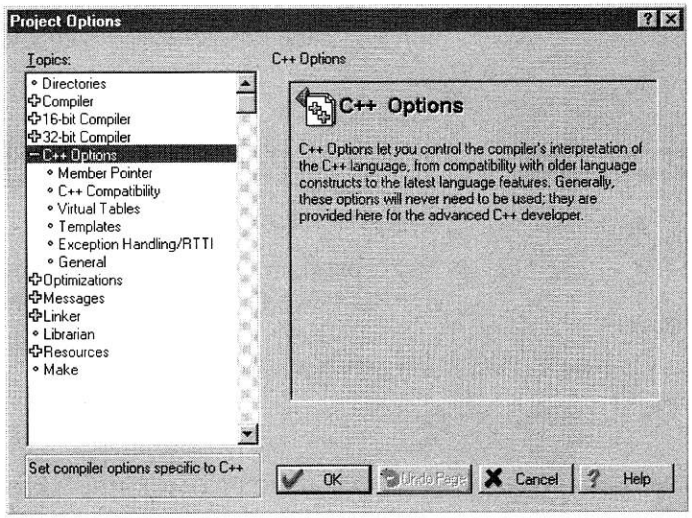
C and C++ languages

See the *Quick Reference* at the end of this booklet.

In Help, find "C++ Options".

You can selectively apply many C++ language constructs by using compiler options and choosing the appropriate libraries. For example, you can eliminate exception-handling from the run-time libraries by linking with the NOEH libraries. You can also use a compiler option to avoid generating run-time type identification. If three distinct character types are a problem with existing code, you can use a compiler option and the appropriate header files to revert to the old behavior.

The Project Options dialog box lets you set C++ options.



Language extensions

In the Help index, search for "C++ keywords".

BORLAND C++ EXTENDS THE LANGUAGE definition by adding keywords that are not part of either the ANSI C Standard or the C++ language. These keywords are options that work within the syntax of the C and C++ languages.

Borland C++ lets you export and import templates in DLLs and EXEs using **__export** so you can streamline your code.

In the Help index, search for "C++ language".

Because C and C++ are machine independent, your machine's resources might not be used efficiently. Borland C++ language extensions let you use machine resources more efficiently. For example, the **__try** and **__finally** keywords are C language extensions that can handle system signals to indicate program exceptions. You can also use **__far**, **__near**, and **__huge** to make better use of segmented memory.

You can use a compiler option to disable all extensions and accept only code that complies with the ANSI Standard.

Writing portable programs

Borland C++ provides tools that can target multiple platforms from a single environment. Borland C++ supports all DOS and Windows operating systems and APIs based on Intel PC processors. DOS applications address these processors in real mode; Windows applications address them in protected mode.

Migrating applications

In the Help index, search for "DOS applications".

BORLAND C++ PROVIDES MIGRATION PATHS across platforms. Borland C++ is designed to minimize the amount of work required to update DOS or Windows applications for use with one of the newer versions of DOS or Windows.

In the Help index, search for "EasyWin".

There is no need to leave your existing DOS code behind. EasyWin simulates DOS in a window that lets you run standard input/output C and C++ programs as Windows applications. Write handy utilities without the programming overhead of *WinMain* and message loops. EasyWin is a great way to experiment with state-of-the-art C and C++ features supported by Borland C++.

Windows applications

Borland simplifies migrating from one Windows version to the next. With Borland C++, you can:

See the *ObjectWindows Programmer's Reference*.

- Add OLE now. The ObjectComponents Framework adds OLE with minimal changes to your code.

In Help, right-click OWL or OCF.

- Change ObjectWindows applications from 16-bit Windows to 32-bit Windows for Windows NT or Windows 95 with the flip of a switch. Just rebuild and run.

In the Help index, search for "VBX".

- Take your VBX controls with you. Even though VBX controls are 16-bit technology, Borland lets you plug them into 32-bit programs including programs running on Windows NT.

DOS applications

Real-mode DOS applications developed with Borland C++ don't need to be recompiled to run as a console-mode application on Windows NT or Windows 95. Most real-mode DOS applications can be recompiled without modification to run as EasyWin applications.

Borland C++ provides lots of tools to develop new or upgrade existing DOS applications. BCC32, the 32-bit compiler, creates 32-bit applications that run in console mode on Windows NT or Windows 95. BCC32i, an additional 32-bit compiler, allows you to optimize your code and produce faster executable code. BCC, the 16-bit compiler, creates 16-bit DOS real-mode applications.

You can also develop DOS applications from the IDE by using TargetExpert.

See the *DOS Reference*.

If your applications are already using all available conventional memory, you can continue to add to your program by using overlays. Large overlaid programs can exceed the 640K DOS memory limitation by swapping code segments to the disk.

Developing international applications

See the *Library Reference*.

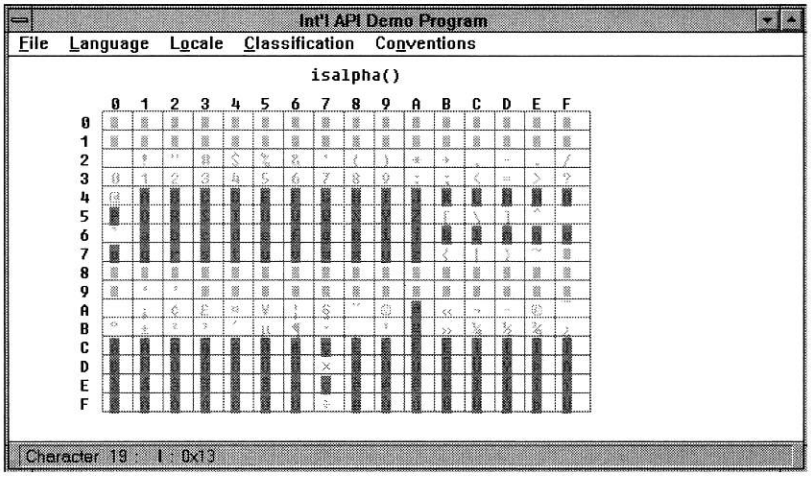
TO DISTRIBUTE YOUR APPLICATIONS to more than one locale, Borland C++ provides support for international software development. In addition to the default locale, you can link run-time library support for the following:

- German
- French
- English (Great Britain)
- English (United States)

In the Help index, search for "locale" and "language".

The results of selecting a locale can be seen by using the INTLDEMO example. The example shows you what characters are processed as alphabetical, digit, space, and so on, according to the ANSI functions. The example also shows how locale specification determines the sort order of file names.

See the complete example in EXAMPLES\OWL\OWLAPPS\INTLDEMO. See also DOC\INTLDEMO.TXT.



Learning more about Borland C++

Information on Borland C++ is available in many forms to help you learn more about the product.

Making online Help work for you

Choose Help from the IDE main menu or press F1 anywhere in the IDE.

BORLAND C++ ONLINE HELP offers an abundance of programming assistance. Whether you need Help on a class or function, want to figure out what an error message means, or are curious about a dialog box setting, press F1 for instant information from the IDE. You can even get Help on both the 16- and 32-bit Windows APIs.

The Help menu lets you access the Help system which provides information on virtually all aspects of the Borland C++ language, libraries, IDE options, compiler options, and any tasks you need to perform while developing C++ applications.

The Help system is organized into Help files that reflect the structure of the documentation set. New contents screens show how the online documents are organized and provide easy access.

Learning by example

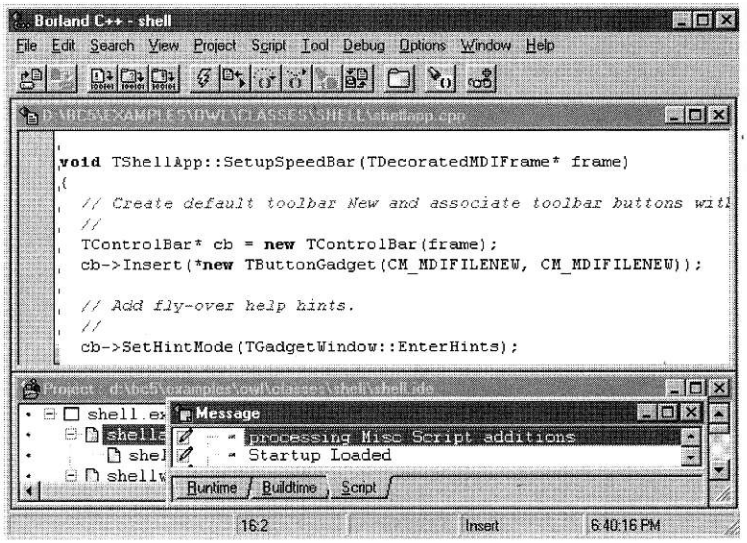
Look in the \BC5\EXAMPLES directories online.

OVER 100 SAMPLE PROGRAMS show how to accomplish common programming tasks using Borland tools. Samples are organized in directories according to topics, such as Windows, ObjectScripting, ObjectComponents, ObjectWindows, and DOS programming.

Browse in the EXAMPLES directory to view examples of the latest features. Try compiling and linking the examples to see how they work.

Learning more about Borland C++

In online Help, you can also find a myriad of examples of APIs, run-time library functions, and resource scripts, many of which you can cut and paste directly into your application.



Reading Borland C++ printed documentation

Refer to the enclosed order form for how to purchase the Borland C++ books or call SAM's Publishing at 800-428-5331 and ask for the set (code: BOR1).

IF YOU PREFER PRINTED MANUALS, all the Borland C++ books are available in printed form. Together with this *Quick Tour & Quick Reference*, the *ObjectScripting Programmer's Guide*, and the *ObjectWindows Quick Reference* poster, you have all the information you need to develop applications with Borland C++.

Watching videos

Order videos from your Borland representative, or call Customer Service at 800-682-9299 (in the US).

IF YOU'D RATHER, you can take a class at your own convenience. Borland's Video Series offers two C++ videos to introduce you to C++ programming:

- *World of C++* —Moving from C to C++
- *Beyond the World of C++* —Intermediate to advanced C++

Getting support

FOR MORE HELP, the Borland Assist program offers technical support plans to suit different users' needs, from individual consultants to large corporations. For details, see the *Borland Assist Support and Services Guide* included with this product.

For more information...

To order these products, contact your Borland representative or call Customer Service at 800-682-9299, ext. 1031 (in the US).

Expanding the power of Borland C++

Borland C++ is the core of a large family of products that together fill the needs of any serious application developer. With additional products, add-ons, and extensions, Borland gives you the tools you need to build robust, state-of-the-art, portable applications for just about any PC platform.

Tools made for developers

Ask for the Borland C++ Development Suite 5.0.

BORLAND OFFERS THE BORLAND C++ Development Suite 5.0 as a complement to Borland C++. It includes the CodeGuard run-time debugger, PVCS version control software, and InstallShield Express, a redeployment tool that helps you collect the right files for running your distributed application. These development tools are all integrated into the IDE.

Writing bug-free code faster

Ask for Borland CodeGuard or purchase the Borland C++ Development Suite 5.0.

IF YOU WANT HELP LOCATING difficult-to-find memory and run-time errors, use CodeGuard. CodeGuard detects memory overruns, reports invalid data references, locates uninitialized data access, and tracks down memory leaks in both 32-bit or 16-bit Windows applications. CodeGuard even catches errors in user code.

This innovative tool helps you decrease development time by automatically checking return values and validating parameters to run-time libraries and Windows API calls. You can get detailed information about detected bugs including error type, source code line, and a stack trace.

Many of the errors that CodeGuard detects are run-time errors that are not caught by the compiler because they do not violate language syntax. These are the types of errors that are likely to cause data corruption while your applications are running. You can also control what information you want CodeGuard to track and report.

Putting VBX controls in your Windows programs

Ask for the Borland Visual Solutions Pack.

TO TAKE ADVANTAGE OF MANY of the prepackaged VBX controls available today, Borland combined many of the industry's most popular and versatile controls together in one box.

The Borland Visual Solutions Pack is a collection of more than 30 drop-in components that you can integrate into your Windows application, whether you've built it with Borland C++, Visual dBASE,® Visual Basic, Visual C++, or any other application development tool that lets you use VBX controls. And only Borland C++ lets you use 16-bit VBX controls in your 32-bit applications.

Expanding the power of Borland C++

The Visual Solutions Pack includes controls for ODBC database support, spreadsheets, WYSIWYG word processors, 3-D and 2-D charts, image editors, serial communications interfaces, notebook tabs, animated buttons, gauges, sliders, and a host of fun gadgets.

Developing Windows Help

Ask for the ForeHelp Help authoring tool.

TO SIMPLIFY YOUR DEVELOPMENT OF Windows Help systems, Borland provides ForeHelp. ForeHelp is a Windows development tool that helps you create Windows Help files without having to understand RTF file formats and without having to interact with the Windows Help compiler. ForeHelp provides a visual development environment that features a full-featured WYSIWYG word processor, as well as complete tools to manage your Help projects.

Developing for OS/2

Ask for Borland C++ for OS/2 with ObjectWindows.

IF YOU DEVELOP DIRECTLY FOR OS/2, OR ARE interested in moving your Windows applications to OS/2, use Borland C++ for OS/2. The most complete C++ development environment for OS/2, the Presentation Manager-hosted IDE includes the tools you need to edit, compile, debug, and run your OS/2 applications quickly. ObjectWindows for OS/2 helps you take your existing ObjectWindows applications to OS/2 with a minimum of effort.

Writing low-level routines

Ask for Turbo Assembler.

THE BORLAND TURBO ASSEMBLER IS A MULTI-PASS assembler that runs on IBM PC computers and compatibles and generates instructions for the 8086, 8088, 80286, 80386, i486, and Pentium, along with 8087, 80287, and 387 numeric coprocessors. This fast, multi-pass assembler provides an interface between C, C++, Pascal, FORTRAN, and COBOL; and 32-bit model and stack frame support. You can link 16- and 32-bit Borland C++ modules with Turbo Assembler modules and produce fast, executable files.

Editing your programs

Ask for the BRIEF editor.

THE BRIEF FULL-FEATURED EDITOR PROVIDES you with the features you need to write professional applications productively in DOS. The powerful macro language lets you customize your programming environment, and its SETUP program ensures BRIEF works instantly with your software. BRIEF provides a host of powerful and flexible features including bookmarks, search and replace, column blocking, 300 levels of undo/redo, and more.

Quick Reference

32-bit compiler (BCC32.EXE and BCC32i.EXE) options *32*

TLINK32 options *35*

TLINK options *36*

16-bit compiler (BCC.EXE) options *37*

16- and 32-bit compiler options *42*

Enable warnings that are Off by default *47*

Disable warnings that are On by default *47*

MAKE options *48*

Library and startup files for 32-bit Windows applications *48*

Library and startup files for DOS applications *49*

Library and startup files for statically linked 16-bit Windows .EXEs *49*

Library and startup files for statically linked 16-bit Windows .DLLs *49*

Library and startup files for dynamically linked 16-bit Windows .EXEs and .DLLs *49*

32-bit compiler options

The following tables summarize the command-line options for the Borland C++ compilers, linkers, and MAKE utility. In the compiler options tables, an asterisk (*) next to an option indicates that the option is on by default.

You'll find a detailed reference for the compiler and linker options in the *Borland C++ User's Guide* (BCW.HLP) online Help file; use the Help index to find details on a specific option. You can find details on the MAKE options in the Borland C++ Tools and Utilities (BCTOOLS.HLP) Help file.

32-bit compiler (BCC32.EXE and BCC32i.EXE) options

Option		Description
@*filename*		Reads compiler options from the response file *filename*
+*filename*		Uses the alternate configuration file *filename*
-3	*	Generates 80386 protected-mode compatible instructions
-4		Generates 80486 protected-mode compatible instructions
-5		Generates Pentium protected-mode compatible instructions
-6		Generates Pentium Pro protected-mode compatible instructions (BCC32i.EXE only)
-A		ANSI language compliance (disables Borland keywords and extensions)
-A-	*	Borland C++ language compliance
-AT		Borland C++ language compliance
-AK		Kernighan and Ritchie language compliance
-AU		UNIX V language compliance
-a-		Does not do any data alignment (same as **-a1**)
-a1	*	Aligns data on byte boundaries
-a2		Aligns data on word boundaries
-a4		Aligns data on double-word boundaries
-a8		Aligns data on quad-word boundaries
-a16		Aligns data on paragraph boundaries
-B		Compiles to assembler (**-S**) and calls the assembler to create an .OBJ file
-b	*	Makes **enums** always integer-sized
-b-		Makes **enums** byte-sized when possible
-C		Turns nested comments on
-C-	*	Turns nested comments off
-c		Compiles to .OBJ but does not link
-D*name*		Defines *name* to the empty string
-D*name=string*		Defines the symbol *name* as *string*
-d		Merges duplicate strings
-d-	*	Does not merge duplicate strings
-E*filename*		Uses *filename* as the assembler
-e*filename*		Links to produce *filename*
-f	*	Allows floating point
-f-		Floating point not supported
-ff	*	Fast floating point (BCC32.EXE only)
-ff-		Strict ANSI floating point (BCC32.EXE only)

32-bit compiler (BCC32.EXE and BCC32i.EXE) options (continued)

Option	Description
-fp	Software workaround for Pentium **fdiv** flaw
-G	Optimizes code for speed (Backward compatibility switch; use **-O2** instead)
-G-	Optimizes code for size (Backward compatibility switch; use **-O1** instead)
-*gn*	Warnings: stop after *n* messages (100 by default)
-H	Generates and uses precompiled headers
-H-	* Does not generate or use precompiled headers
-Hc	Cache precompiled headers; must be used with **-H** or **-H"xxx"**
-Hu	Uses but does not generate precompiled headers
-H"*xxx*"	Stops compiling precompiled headers at file "*xxx*"
-H=*filename*	Sets the name of the file for precompiled headers
-I*path*	Sets search path for directories for include files
-i*n*	Makes significant identifier length to be *n* (default = 250)
-Jg	* Generates definitions for all template instances and merges duplicates
-Jgd	Generates public definitions for all template instances
-Jgx	Generates external references for all template instances
-j*n*	Errors: stop after *n* messages (25 messages by default)
-K	Default character type unsigned
-K-	* Default character type signed
-k	* Enables the standard stack frame
-L*path*	Sets search path for library files to *path*
-l*x*	Passes option *x* to the linker (can use more than one *x*)
-l-*x*	Disables option *x* for the linker
-M	Instructs the linker to create a map file
-n*path*	Sets the output directory to *path*
-O	Optimizes jumps
-O1	Generates smallest possible code by setting the following switches: **-O**, **-Oc**, **-O1**, and **-OS**
-O2	Generates fastest possible code by setting the following switches: **-Oc**, **-Oi**, **-O1**, **-OS**, and **-Ov**
-Oc	Eliminates duplicate expressions within basic blocks (BCC32.EXE only) (BCC32.EXE treats **-Oc** and **-Og** the same)
-Od	Disables all optimizations (BCC32.EXE only)
-Og	Eliminates duplicate expressions within functions (BCC32.EXE only) (BCC32.EXE treats **-Oc** and **-Og** the same)
-OI	Optimizes across function boundaries (BCC32i.EXE only)
-Oi	Expands common intrinsic functions inline (BCC32.EXE only)
-OM	Cache hit optimizations (BCC32i.EXE only)
-OS	Instruction scheduling
-O-S	Disables instruction scheduling
-Os	Generates smallest possible code (Backward compatibility switch; use **-O1** instead)
-Ot	Generates fastest possible code (Backward compatibility switch; use **-O2** instead)
-Ov	Enables loop induction variable and strength reduction (BCC32.EXE only)
-Ox	Generates fastest possible code (Backward compatibility switch; use **-O2** instead)
-o*filename*	Compiles source file to *filename* .OBJ
-P	Performs a C++ compile regardless of source file extension

32-bit compiler (BCC32.EXE and BCC32i.EXE) options (continued)

Option		Description
-P-		Performs a C++ compile depending on source file extension
-P*ext*		Performs a C++ compile and sets the default extension to *ext*
-p		Uses **pascal** calling convention
-p-		Uses **cdecl** calling convention
-pc	*	Uses **cdecl** calling convention
-pr		Uses **fastcall** calling convention for passing parameters in registers
-ps		Uses **stdcall** calling convention
-R		Includes browser information in generated .OBJ files
-RT	*	Enables RTTI (run-time type identification)
-r	*	Uses register variables
-r-		Disables the use of register variables
-S		Produces .ASM output file
-T*x*		Passes *x* as an option to the assembler specified with **-E**
-T-		Removes all previous assembler options
-tWM		Generates a multithreaded executable
-U*name*		Undefines any previous definitions of *name*
-u	*	Generates underscores
-V	*	Uses smart C++ virtual tables
-V0		External C++ virtual tables
-V1		Public C++ virtual tables
-VC		Calling convention mangling compatibility (Backward compatibility switch)
-Vd		Does not restrict scope of **for** loop expression variables (Backward compatibility switch)
-Ve		Allows zero-length empty base classes
-VF		Enables Microsoft foundation class (MFC) compatibility compilation
-v		Generates debugging information; also sets **-vi-**
-v-		No debugging information
-vi		Expands inline functions inline
-vi-		Expands inline functions out of line
-WM		Generates a multithreaded executable (Backward compatibility switch; use **-tWM** instead)
-w		Enables the display of warnings
-w-		Disables the display of all warnings
-w*xxx*		Enables *xxx* warning message (See page 47 for **-w** message details)
-w-*xxx*		Disables *xxx* warning message
-w!		Returns a nonzero code from the command-line compiler when there are warnings
-X		Disables compiler autodependency output
-X-		Uses compiler autodependency output
-x	*	Enables exception handling
-xc		Enables compatible exceptions
-xd		Enables destructor cleanup
-xf		Enables fast exception prologs
-xp		Enables exception location information
-y		Generates source line numbers for debugging

TLINK32 options

Option	Description
@*xxxx*	Uses the response file *xxxx*
/A:*dd*	Specifies file alignment (Backward compatibility switch; use **/Af**)
/Af:*nnnn*	Specifies file alignment; set *nnnn* in hex or decimal (0x200 = 512-byte boundaries)
/Ao:*nnnn*	Specifies object alignment; set *nnnn* in hex or decimal (0x1000 = 4096-byte boundaries)
/aa	Builds a 32-bit Window application
/ap	Builds a 32-bit Windows console application that can be run in a window
/B:*xxxx*	Specifies image base address
/c	Treats case as significant in symbols
/E*nn*	Specifies maximum errors before termination
/H:*xxxx*	Specifies application heap reserve size in hexadecimal
/Hc:*xxxx*	Specifies application heap commit size in hexadecimal
/j	Specifies object search paths
/L	Specifies library search paths
/M	Mangled names in map
/m	Creates map file with public names
/n	No default libraries
/o	Imports by ordinals
/P-	Disables code packing
/r	Verbose link
/S:*xxxx*	Specifies application stack reserve size in hexadecimal
/Sc:*xxxx*	Specifies application stack commit size in hexadecimal
/s	Detailed map of segments
/Tpd	Targets a Windows .DLL file
/Tpe	Targets a Windows .EXE file
/V*d.d*	Specifies expected Windows version
/v	Includes full symbolic debug information
/wdef	Enables "No .DEF file" warning (warning is Off by default)
/wdpl	Enables "Duplicate symbol in library" warning (warning is Off by default)
/wimt	Enables "Import does not match previous definition" warning (warning is Off by default)
/wmsk	Enables "Multiple stack segments" warning (warning is Off by default)
/w-bdl	Disables "Using based linking in .DLL" warning (warning is On by default)
/w-dll	Disables ".EXE module built with a .DLL extension" warning (warning is On by default)
/w-dup	Disables "Duplicate symbol" warning (warning is On by default)
/w-ent	Disables "No entry point" warning (warning is On by default)
/w-inq	Disables "Extern not qualified with __import" warning (warning is On by default)
/w-srf	Disables "Self-relative fixup overflowed" warning (warning is On by default)
/w-stk	Disables "No stack" warning (warning is On by default)
/x	Surpresses creation of map file

TLINK options

Option	Description
@*xxxx*	Uses the response file *xxxx*
/3	Enables 32-bit processing
/A=*dd*	Sets segment alignment; set ***dd*** to a decimal value between 2 and 65,535
/C	Treats case as significant in EXPORTS and IMPORTS section of module-definition file
/c	Treats case as significant in symbols
/d	Warns if duplicate symbols in libraries
/f	Inhibits optimization of far calls to near data
/Gn	Discards nonresident name table
/Gr	Transfers resident names to nonresident names table
/i	Initializes all segments
/j	Specifies object search paths
/k	Suppresses "No stack" warning messages
/L	Specifies library search paths
/l	Includes source line numbers in map file
/M	Creates a map file with mangled public names
/m	Creates map file with public names
/n	Ignores default libraries
/Oa	Minimum segment alignment (optimization)
/Oc	Chained fixups (optimization)
/Oi	Iterated data (optimization)
/Or	Minimum resource alignment (optimization)
/o	Overlays modules or libraries (DOS only)
/P[=dd]	Packs code segments
/R[mpekv]	Specifies option to RLINK
/s	Creates a map file with a detailed map of segments
/Tdc	Targets a tiny-model DOS .COM file
/Tde	Targets DOS .EXE file
/Twd	Targets Windows .DLL file
/Twe	Targets Windows .EXE file
/Txe	Targets DPMI .EXE file
/t	Creates a DOS .COM file (same as **/Tdc**)
/V*d.d*	Specifies expected Windows version
/v	Includes full symbolic debug information in nonresident names table
/x	Surpresses creation of map file
/ye	Uses expanded memory for swapping
/yx	Uses extended memory for swapping

16-bit compiler (BCC.EXE) options

Option		Description
@*filename*		Reads compiler options from the response file *filename*
+*filename*		Uses the alternate configuration file *filename*
-1-		Generates 8086 instructions
-1		Generates 80186 instructions
-2	*	Generates 80286 protected-mode compatible instructions
-3		Generates 80386 protected-mode compatible instructions
-4		Generates 80486 protected-mode compatible instructions
-A		ANSI language compliance (disables Borland keywords and extensions)
-A-	*	Borland C++ language compliance
-AT		Borland C++ language compliance
-AK		Kernighan and Ritchie language compliance
-AU		UNIX V language compliance
-a-	*	Does not do any data alignment (same as **-a1**)
-a1		Aligns data on byte boundaries
-a2		Aligns data on word boundaries
-B		Compiles to assembler (**-S**) and calls the assembler to create an .OBJ file
-b	*	Makes **enums** always integer-sized
-b-		Makes **enums** byte-sized when possible
-C		Turns nested comments on
-C-	*	Turns nested comments off
-c		Compiles to .OBJ but does not link
-D*name*		Defines *name* to the empty string
-D*name=string*		Defines the symbol *name* as *string*
-d		Merges duplicate strings
-d-	*	Does not merge duplicate strings
-dc		Moves string literals from data segment to code segment
-E*filename*		Uses *filename* as the assembler
-e*filename*		Links to produce *filename*
-Fc		Generates COMDEFs
-Ff		Creates far data automatically
-Ff=*size*		Creates far data for global data items that are >= *size* (in bytes)
-Fm		Enables the **-Fc**, **-Ff**, and **-Fs** options
-Fs		Assumes DS = SS in all memory models (DOS only)
-f	*	Emulate floating point
-f-		Floating point not supported
-ff	*	Fast floating point
-ff-		Strict ANSI floating point
-fp		Software workaround for Pentium **fdiv** flaw
-f87		Uses 8087 hardware instructions (DOS only)
-f287		Uses 80287 hardware instructions (DOS only)
-G		Optimizes code for speed (Backward compatibility switch; use **-O2** instead)
-G-		Optimizes code for size (Backward compatibility switch; use **-O1** instead)
-g*n*		Warnings: stop after *n* messages (100 by default)
-H		Generates and uses precompiled headers

16-bit compiler (BCC.EXE) options (continued)

Option	Description
-H-	* Does not generate or use precompiled headers
-Hc	Cache precompiled headers; must be used with **-H** or **-H"xxx"**
-Hu	Uses but does not generate precompiled headers
-H"*xxx*"	Stops compiling precompiled headers at file "*xxx*"
-H=*filename*	Sets the name of the file for precompiled headers
-h	Uses fast huge pointer arithmetic
-I*path*	Sets search path for directories for include files
-i*n*	Makes significant identifier length to be *n* (default = 250)
-Jg	* Generates definitions for all template instances and merges duplicates
-Jgd	Generates public definitions for all template instances
-Jgx	Generates external references for all template instances
-j*n*	Errors: stop after *n* messages (25 messages by default)
-K	Default character type unsigned
-K-	* Default character type signed
-K2	Allows only two character types (unsigned and signed); char is treated as signed (Backward compatibility switch; Borland C++ 3.1 and earlier)
-k	* Enables the standard stack frame
-L*path*	Sets search path for library files to *path*
-l*x*	Passes option *x* to the linker (can use more than one *x*)
-l-*x*	Disables option *x* for the linker
-M	Instructs the linker to create a map file
-mc	Compiles using compact memory model
-mc!	Compiles using compact memory model; assume DS != SS
-mh	Compiles using huge memory model (DOS only)
-mh!	Compiles using huge memory model; assume DS != SS (DOS only)
-ml	Compiles using large memory model
-ml!	Compiles using large memory model; assume DS != SS
-mm	Compiles using medium memory model
-mm!	Compiles using medium memory model; assume DS != SS
-ms	* Compiles using small memory model
-ms!	Compiles using small memory model; assume DS != SS
-mt	Compiles using tiny memory model (DOS only)
-mt!	Compiles using tiny memory model; assume DS != SS (DOS only)
-N	Checks for stack overflow
-n*path*	Sets the output directory to *path*
-O	* Optimizes jumps
-O1	Generates smallest possible code by setting the following switches: **-O**, **-Ob**, **-Oc**, **-Oe**, **-O1**, and **-Z**
-O2	Generates fastest possible code by setting the following switches: **-Ob**, **-Oe**, **-Og**, **-Oi**, **-O1**, **-Om**, **-Op**, **-Ov**, and **-Z**
-Oa	Optimizes code assuming pointer expressions are not aliased on common subexpression evaluation
-Ob	Eliminates dead code
-Oc	Eliminates duplicate expressions within basic blocks
-Od	Disables all optimizations

16-bit compiler (BCC.EXE) options (continued)

Option	Description
-Oe	Registers allocation and live range analysis
-Og	Eliminates duplicate expressions within functions
-Oi	Expands common intrinsic functions inline
-Ol	Optimizes loops
-Om	Moves invariant code out of loops
-Op	Propagates copies
-Os	Generates smallest possible code (Backward compatibility switch; use **-O1** instead)
-Ot	Generates fastest possible code (Backward compatibility switch; use **-O2** instead)
-Ov	Enables loop induction variable and strength reduction
-OW	Suppresses the **inc bp/dec bp** on Windows far functions
-Ox	Generates fastest possible code (Backward compatibility switch; use **-O2** instead)
-o*filename*	Compiles source file to *filename* .OBJ
-P	Performs a C++ compile regardless of source file extension
-P-	Performs a C++ compile depending on source file extension
-P*ext*	Performs a C++ compile and sets the default extension to *ext*
-p	Uses **pascal** calling convention
-p-	Uses **cdecl** calling convention
-pc	* Uses **cdecl** calling convention
-po	Uses **fastthis** calling convention for passing **this** parameter in registers
-pr	Uses **fastcall** calling convention for passing parameters in registers
-R	Includes browser information in generated .OBJ files
-RT	* Enables RTTI (run-time type identification)
-r	* Uses register variables
-r-	Disables the use of register variables
-rd	Allows only declared register variables to be kept in registers
-S	Produces .ASM output file
-T*x*	Passes *x* as an option to TASM or to the assembler specified with **-E**
-T-	Removes all previous assembler options
-tD	Generates a DOS .EXE file
-tDc	Generates a DOS .COM file
-tDe	Generates a DOS .EXE file
-tW	Generates a Windows .EXE with all functions exportable
-tWC	Generates a console .EXE with all functions exportable
-tWCD	Generates a console .DLL with all functions exportable
-tWCDE	Generates a console .DLL with explicit functions exportable
-tWD	Generates a Windows .DLL with all functions exportable
-tWDE	Generates a Windows .DLL with explicit functions exportable
-tWE	Generates a Windows .EXE with explicit functions exportable
-tWS	Generates a Windows .EXE that uses smart callbacks with all functions exportable
-tWSE	Generates a Windows .EXE that uses smart callbacks with explicit functions exportable
-U*name*	Undefines any previous definitions of *name*
-u	* Generates underscores
-V	* Uses smart C++ virtual tables
-V0	External C++ virtual tables

16-bit compiler (BCC.EXE) options (continued)

Option	Description
-Vl	Public C++ virtual tables
-Va	Passes class arguments by reference to a temporary variable (Backward compatibility switch)
-Vb	Makes virtual base class pointer same size as **this** pointer of the class (Backward compatibility switch)
-Vb-	* Makes virtual base class pointer always near
-VC	Calling convention mangling compatibility (Backward compatibility switch)
-Vc	Does not add hidden members and code to derived classes (Backward compatibility switch)
-Vd	Does not restrict scope of **for** loop expression variables (Backward compatibility switch)
-Ve	Allows zero-length empty base classes
-VF	Enables Microsoft foundation class (MFC) compatibility compilation
-Vf	Far C++ virtual tables
-Vh	Treats "far" class as "huge"
-Vmd	Uses the smallest representation for member pointers
-Vmm	Member pointers support multiple inheritance
-Vmp	Honors the declared precision for all member pointer types
-Vms	Member pointers support single inheritance
-Vmv	* Member pointers have no restrictions (most general representation)
-Vo	Enables the backward compatibility options **-Va**, **-Vb**, **-Vc**, **-Vp**, **-Vt**, and **-Vv**
-Vp	Passes the **this** parameter to **pascal** member functions as the first parameter on the stack (Backward compatibility switch)
-Vs	Local C++ virtual tables
-Vt	Places the virtual table pointer after nonstatic data members (Backward compatibility switch)
-Vv	Does not add the hidden members and code to classes with pointers to virtual base class members (Backward compatibility switch)
-v	Generates debugging information; also sets **-vi-**
-v-	No debugging information
-vi	Expands inline functions inline
-vi-	Expands inline functions out of line
-W	Generates a Windows .EXE with all functions exportable (Backward compatibility switch; use **-tW** instead)
-WD	Generates a Windows .DLL with all functions exportable (Backward compatibility switch; use **-tWD** instead)
-WDE	Generates a Windows .DLL with explicit functions exportable (Backward compatibility switch; use **-tWDE** instead)
-WE	Generates a Windows .EXE with explicit functions exportable (Backward compatibility switch; use **-tWE** instead)
-WS	Generates a Windows .EXE that uses smart callbacks with all functions exportable (Backward compatibility switch; use **-tWS** instead)
-WSE	Generates a Windows .EXE that uses smart callbacks, with explicit functions exportable (Backward compatibility switch; use **-tWDE** instead)
-WX	Generates a 16-bit DPMI executable (DOS only)
-w	Enables the display of warnings
-w-	Disables the display of all warnings
-w*xxx*	Enables *xxx* warning message (See page 47 for **-w** message details)
-w-*xxx*	Disables *xxx* warning message
-w!	Returns a nonzero code from the command-line compiler when there are warnings

16-bit compiler (BCC.EXE) options (continued)

Option	Description
-X	Disables compiler autodependency output
-X-	Uses compiler autodependency output
-x	* Enables exception handling
-xc	Enables compatible exceptions
-xd	Enables destructor cleanup
-xf	Enables fast exception prologs
-xp	Enables exception location information
-Y	Enables overlay code generation (DOS only)
-Yo	Overlays the compiled files (DOS only)
-y	Generates source line numbers for debugging
-Z	Enables register load suppression optimization
-zA*name*	Uses *name* for code class
-zB*name*	Uses *name* for BSS class
-zC*name*	Uses *name* for code segment
-zD*name*	Uses *name* for BSS segment
-zE*name*	Uses *name* for far segment
-zF*name*	Uses *name* for far class
-zG*name*	Uses *name* for BSS group
-zH*name*	Uses *name* for far group
-zP*name*	Uses *name* for code group
-zR*name*	Uses *name* for data segment
-zS*name*	Uses *name* for data group
-zT*name*	Uses *name* for data class
-zV*name*	Uses *name* for far virtual segment
-zW*name*	Uses *name* for far virtual class
-zA*	Uses default name for code class
-zB*	Uses default name for BSS class
-zC*	Uses default name for code segment
-zD*	Uses default name for BSS segment
-zE*	Uses default name for far segment
-zF*	Uses default name for far class
-zG*	Uses default name for BSS group
-zH*	Uses default name for far group
-zP*	Uses default name for code group
-zR*	Uses default name for data segment
-zS*	Uses default name for data group
-zT*	Uses default name for data class
-zV*	Uses default name for far virtual segment
-zW*	Uses default name for far virtual class

16- and 32-bit compiler options

Option	Platform		Description
@*filename*	16	32	Reads compiler options from the response file *filename*
+*filename*	16	32	Uses the alternate configuration file *filename*
-1-	16		Generates 8086 instructions
-1	16		Generates 80186 instructions
-2	16*		Generates 80286 protected-mode compatible instructions
-3	16	32*	Generates 80386 protected-mode compatible instructions
-4	16	32	Generates 80486 protected-mode compatible instructions
-5		32	Generates Pentium protected-mode compatible instructions
-6		32	Generates Pentium Pro protected-mode compatible instructions (BCC32i.EXE only)
-A	16	32	ANSI language compliance (disables Borland keywords and extensions)
-A-, AT	16*	32*	Borland C++ language compliance
-AK	16	32	Kernighan and Ritchie language compliance
-AU	16	32	UNIX V language compliance
-a1, -a-	16*	32*	Aligns data on byte boundaries
-a2	16	32	Aligns data on word boundaries
-a4		32	Aligns data on double-word boundaries
-a8		32	Aligns data on quad-word boundaries
-a16		32	Aligns data on paragraph boundaries
-B	16	32	Compiles to assembler (**-S**) and calls the assembler to create an .OBJ file
-b	16*	32*	Makes **enums** always integer-sized
-b-	16	32	Makes **enums** byte-sized when possible
-C	16	32	Turns nested comments on
-C-	16*	32*	Turns nested comments off
-c	16	32	Compiles to .OBJ but does not link
-D*name*	16	32	Defines *name* to the empty string
-D*name=string*	16	32	Defines the symbol *name* as *string*
-d	16	32	Merges duplicate strings
-d-	16*	32*	Does not merge duplicate strings
-dc	16		Moves string literals from data segment to code segment
-E*filename*	16	32	Uses *filename* as the assembler
-e*filename*	16	32	Links to produce *filename*
-Fc	16		Generates COMDEFs
-Ff	16		Creates far data automatically
-Ff=*size*	16		Creates far data for global data items that are >= *size* (in bytes)
-Fm	16		Enables the **-Fc**, **-Ff**, and **-Fs** options
-Fs	16		Assumes DS = SS in all memory models (DOS only)
-f	16*	32*	Emulate floating point
-f-	16	32	Floating point not supported
-ff	16*	32*	Fast floating point
-ff-	16	32	Strict ANSI floating point
-fp	16	32	Software workaround for Pentium **fdiv** flaw
-f87	16		Uses 8087 hardware instructions (DOS only)
-f287	16		Uses 80287 hardware instructions (DOS only)
-G	16	32	Optimizes code for speed (Backward compatibility switch; use **-O2** instead)

16- and 32-bit compiler options (continued)

Option	Platform		Description
-G-	16	32	Optimizes code for size (Backward compatibility switch; use **-O1** instead)
-g*n*	16	32	Warnings: stop after *n* messages (100 by default)
-H	16	32	Generates and uses precompiled headers
-H-	16*	32*	Does not generate or use precompiled headers
-Hc	16	32	Cache precompiled headers; must be used with **-H** or **-H"xxx"**
-Hu	16	32	Uses but does not generate precompiled headers
-H"*xxx*"	16	32	Stops compiling precompiled headers at file "*xxx*"
-H=*filename*	16	32	Sets the name of the file for precompiled headers
-h	16		Uses fast huge pointer arithmetic
-I*path*	16	32	Sets search path for directories for include files
-i*n*	16	32	Makes significant identifier length to be *n* (default = 250)
-Jg	16*	32*	Generates definitions for all template instances and merges duplicates
-Jgd	16	32	Generates public definitions for all template instances
-Jgx	16	32	Generates external references for all template instances
-j*n*	16	32	Errors: stop after *n* messages (25 messages by default)
-K	16	32	Default character type unsigned
-K-	16*	32*	Default character type signed
-K2	16		Allows only two character types (unsigned and signed); char is treated as signed (Backward compatibility switch; Borland C++ 3.1 and earlier)
-k	16*	32*	Enables the standard stack frame
-L*path*	16	32	Sets search path for library files to *path*
-l*x*	16	32	Passes option *x* to the linker (can use more than one *x*)
-l-*x*	16	32	Disables option *x* for the linker
-M	16	32	Instructs the linker to create a map file
-mc	16		Compiles using compact memory model
-mc!	16		Compiles using compact memory model; assume DS != SS
-mh	16		Compiles using huge memory model (DOS only)
-mh!	16		Compiles using huge memory model; assume DS != SS (DOS only)
-ml	16		Compiles using large memory model
-ml!	16		Compiles using large memory model; assume DS != SS
-mm	16		Compiles using medium memory model
-mm!	16		Compiles using medium memory model; assume DS != SS
-ms	16*		Compiles using small memory model
-ms!	16		Compiles using small memory model; assume DS != SS
-mt	16		Compiles using tiny memory model (DOS only)
-mt!	16		Compiles using tiny memory model; assume DS != SS (DOS only)
-N	16		Checks for stack overflow
-n*path*	16	32	Sets the output directory to *path*
-O	16*	32	Optimizes jumps
-O1	16	32	Generates smallest possible code by setting the following switches: **-O**, **-Ob**, **-Oc**, **-Oe**, **-O1**, **-OS**, and **-Z**
-O2	16	32	Generates fastest possible code by setting the following switches: **-Ob**, **-Oc**, **-Oe**, **-Og**, **-Oi**, **-O1**, **-Om**, **-Op**, **-OS**, **-Ov**, and **-Z**
-Oa	16		Optimizes code assuming pointer expressions are not aliased on common subexpression evaluation

16- and 32-bit compiler options (continued)

Option	Platform		Description
-Ob	16		Eliminates dead code
-Oc	16	32	Eliminates duplicate expressions within basic blocks
-Od	16	32	Disables all optimizations
-Oe	16		Registers allocation and live range analysis
-Og	16	32	Eliminates duplicate expressions within functions
-OI		32	Optimizes across function boundaries (BCC32i.EXE only)
-Oi	16	32	Expands common intrinsic functions inline
-Ol	16		Optimizes loops
-OM		32	Cache hit optimizations (BCC32i.EXE only)
-Om	16		Moves invariant code out of loops
-Op	16		Propagates copies
-OS		32	Instruction scheduling
-O-S		32	Disables instruction scheduling
-Os	16	32	Generates smallest possible code (Backward compatibility switch; use **-O1** instead)
-Ot	16	32	Generates fastest possible code (Backward compatibility switch; use **-O2** instead)
-Ov	16	32	Enables loop induction variable and strength reduction
-OW	16		Suppresses the **inc bp/dec bp** on Windows far functions
-Ox	16	32	Generates fastest possible code (Backward compatibility switch; use **-O2** instead)
-o*filename*	16	32	Compiles source file to *filename* .OBJ
-P	16	32	Performs a C++ compile regardless of source file extension
-P-	16	32	Performs a C++ compile depending on source file extension
-P*ext*	16	32	Performs a C++ compile and sets the default extension to *ext*
-p	16	32	Uses **pascal** calling convention
-pc, -p-	16*	32*	Uses **cdecl** calling convention
-po	16	32	Uses **fastthis** calling convention for passing **this** parameter in registers
-pr	16	32	Uses **fastcall** calling convention for passing parameters in registers
-ps		32	Uses **stdcall** calling convention
-R	16	32	Includes browser information in generated .OBJ files
-RT	16*	32*	Enables RTTI (run-time type identification)
-r	16*	32*	Uses register variables
-r-	16	32	Disables the use of register variables
-rd	16		Allows only declared register variables to be kept in registers
-S	16	32	Produces .ASM output file
-T*x*	16	32	Passes *x* as an option to TASM or to the assembler specified with **-E**
-T-	16	32	Removes all previous assembler options
-tD	16		Generates a DOS .EXE file
-tDc	16		Generates a DOS .COM file
-tDe	16		Generates a DOS .EXE file
-tW	16		Generates a Windows .EXE with all functions exportable
-tWC	16		Generates a console .EXE with all functions exportable
-tWCD	16		Generates a console .DLL with all functions exportable
-tWCDE	16		Generates a console .DLL with explicit functions exportable
-tWD	16		Generates a Windows .DLL with all functions exportable
-tWDE	16		Generates a Windows .DLL with explicit functions exportable

16- and 32-bit compiler options (continued)

Option	Platform		Description
-tWE	16		Generates a Windows .EXE with explicit functions exportable
-tWM		32	Generates a multithreaded executable
-tWS	16		Generates a Windows .EXE that uses smart callbacks with all functions exportable
-tWSE	16		Generates a Windows .EXE that uses smart callbacks with explicit functions exportable
-U*name*	16	32	Undefines any previous definitions of *name*
-u	16*	32*	Generates underscores
-V	16*	32*	Uses smart C++ virtual tables
-V0	16	32	External C++ virtual tables
-V1	16	32	Public C++ virtual tables
-Va	16		Passes class arguments by reference to a temporary variable (Backward compatibility switch)
-Vb	16		Makes virtual base class pointer same size as **this** pointer of the class (Backward compatibility switch)
-Vb-	16*		Makes virtual base class pointer always near
-VC	16	32	Calling convention mangling compatibility (Backward compatibility switch)
-Vc	16		Does not add hidden members and code to derived classes (Backward compatibility switch)
-Vd	16	32	Does not restrict scope of **for** loop expression variables (Backward compatibility switch)
-Ve	16	32	Allows zero-length empty base classes
-VF	16	32	Enables Microsoft foundation class (MFC) compatibility compilation
-Vf	16		Far C++ virtual tables
-Vh	16		Treats "far" class as "huge"
-Vmd	16		Uses the smallest representation for member pointers
-Vmm	16		Member pointers support multiple inheritance
-Vmp	16		Honors the declared precision for all member pointer types
-Vms	16		Member pointers support single inheritance
-Vmv	16*		Member pointers have no restrictions (most general representation)
-Vo	16		Enables the backward compatibility options **-Va**, **-Vb**, **-Vc**, **-Vp**, **-Vt**, and **-Vv**
-Vp	16		Passes the **this** parameter to **pascal** member functions as the first parameter on the stack (Backward compatibility switch)
-Vs	16		Local C++ virtual tables
-Vt	16		Places the virtual table pointer after nonstatic data members (Backward compatibility switch)
-Vv	16		Does not add the hidden members and code to classes with pointers to virtual base class members (Backward compatibility switch)
-v	16	32	Generates debugging information; also sets **-vi-**
-v-	16	32	No debugging information
-vi	16	32	Expands inline functions inline
-vi-	16	32	Expands inline functions out of line
-W	16		Generates a Windows .EXE with all functions exportable (Backward compatibility switch; use **-tW** instead)
-WD	16		Generates a Windows .DLL with all functions exportable (Backward compatibility switch; use **-tWD** instead)
-WDE	16		Generates a Windows .DLL with explicit functions exportable (Backward compatibility switch; use **-tWDE** instead)

16- and 32-bit compiler options (continued)

Option	Platform		Description
-WE	16		Generates a Windows .EXE with explicit functions exportable (Backward compatibility switch; use **-tWE** instead)
-WM		32	Generates a multithreaded executable (Backward compatibility switch; use **-tWM** instead)
-WS	16		Generates a Windows .EXE that uses smart callbacks with all functions exportable (Backward compatibility switch; use **-tWS** instead)
-WSE	16		Generates a Windows .EXE that uses smart callbacks, with explicit functions exportable (Backward compatibility switch; use **-tWDE** instead)
-WX	16		Generates a 16-bit DPMI executable (DOS only)
-w	16	32	Enables the display of warnings
-w-	16	32	Disables the display of all warnings
-w*xxx*	16	32	Enables *xxx* warning message (See page 47 for **-w** message details)
-w-*xxx*	16	32	Disables *xxx* warning message
-w!	16	32	Returns a nonzero code from the command-line compiler when there are warnings
-X	16	32	Disables compiler autodependency output
-X-	16	32	Uses compiler autodependency output
-x	16*	32*	Enables exception handling
-xc	16	32	Enables compatible exceptions
-xd	16	32	Enables destructor cleanup
-xf	16	32	Enables fast exception prologs
-xp	16	32	Enables exception location information
-Y	16		Enables overlay code generation (DOS only)
-Yo	16		Overlays the compiled files (DOS only)
-y	16	32	Generates source line numbers for debugging
-Z	16		Enables register load suppression optimization
-zA*name*	16		Uses *name* for code class (**-zA*** uses default name for code class)
-zB*name*	16		Uses *name* for BSS class (**-zB*** uses default name for BSS class)
-zC*name*	16		Uses *name* for code segment (**-zC*** uses default name for code segment)
-zD*name*	16		Uses *name* for BSS segment (**-zD*** uses default name for BSS segment)
-zE*name*	16		Uses *name* for far segment (**-zE*** uses default name for far segment)
-zF*name*	16		Uses *name* for far class (**-zF*** uses default name for far class)
-zG*name*	16		Uses *name* for BSS group (**-zG*** uses default name for BSS group)
-zH*name*	16		Uses *name* for far group (**-zH*** uses default name for far group)
-zP*name*	16		Uses *name* for code group (**-zP*** uses default name for code group)
-zR*name*	16		Uses *name* for data segment (**-zR*** uses default name for data segment)
-zS*name*	16		Uses *name* for data group (**-zS*** uses default name for data group)
-zT*name*	16		Uses *name* for data class (**-zT*** uses default name for data class)
-zV*name*	16		Uses *name* for far virtual segment (**-zV*** uses default name for virtual segment)
-zW*name*	16		Uses *name* for far virtual class (**-zW*** uses default name for virtual class)

Compiler warnings

Enable warnings that are Off by default

Option	Description	Option	Description
-wamb	Ambiguous operators need parentheses	-wnod	No declaration for <ident>
-wamp	Superflous & with function	-wpin	Initialization is only partially bracketed
-wasm	Unkown assembler instruction	-wsig	Conversion may loose significant digits
-wbbf	Bit fields must be signed or unsigned int	-wstu	Undefined structure <ident>
-wcln	Constant is long	-wstv	Structure passed by value
-wdef	Possible use of <ident> before definition	-wucp	Mixing pointers to different 'char' types
-wnak	Non-ANSI keyword used: <ident>	-wuse	<ident> is declared but never used

Disable warnings that are On by default

Option	Description	Option	Description
-w-asc	Restarting compile using assembly	-w-nci	Constant member <ident> is not initialized
-w-aus	<ident> is a signed value that is never used	-w-ncl	Constructor initializer list ignored
-w-bei	Initializing <ident> with <ident>	-w-nfd	Function body ignored
-w-big	Hex value contains more than 3 digits	-w-nin	Initializer for object <ident> ignored
-w-ccc	Condition is always <ident>	-w-nma	Macro definition ignored
-w-com	Continuation character / found in // comment	-w-nmu	#undef directive ignored
-w-cpt	Nonportable pointer comparison	-w-nsf	Declaration of static function <ident> ignored
-w-dig	Declaration ignored	-w-nst	Use qualifed name to access member type <ident>
-w-dpu	Declare <ident> prior to use in prototype	-w-ntd	Use '> >' for nested templates instead of '>>'
-w-dsz	Array size 'delete' ignored	-w-obi	Base initalization without a class name obsolete
-w-dup	Redefinition of <ident> is not identical	-w-obs	<ident> is obsolete
-w-eas	Assigning <ident> to <ident>	-w-ofp	Style of function definition is now obsolete
-w-eff	Code has no effect	-w-par	Parameter <ident> is never used
-w-ext	<ident> is declared as both external and static	-w-pch	Cannot create precompiled header: <ident>
-w-hch	Handler for <ident> hidden by previous handler for <ident>	-w-pck	Structure packing size has changed
-w-hid	<ident> hides virtual function <ident>	-w-pia	Possibly incorrect assignment
-w-ias	Array variable <ident> is near	-w-pre	Overloaded prefix operator <ident> used as a postfix operator
-w-ibc	Base class <ident> is also a base class of <ident>	-w-pro	Call to function with no prototype
-w-ill	Ill-formed pragma	-w-rch	Unreachable code
-w-inl	Functions <ident> are not expanded inline	-w-ret	Both return and return with a value used
-w-lin	Temporary used to initialize <ident>	-w-rng	Constant out of range in comparison
-w-lvc	Temporary used for parameter <ident>	-w-rpt	Nonprotable pointer conversion
-w-mpc	Conversion to <ident> will fail for members of virtual base <ident>	-w-rvl	Function should return a value
-w-mpd	Maximum precision used for member pointer type <ident>	-w-sus	Suspicious pointer conversion
-w-msg	<User-defined message>	-w-voi	void functions may not return a value
-w-ncf	Non-const/volatile function <ident> called for const/volatile object	-w-zdi	Division by zero

MAKE options

Option	Description
-a	Checks dependencies of include files and nested include files associated with .OBJ files and updates the .OBJ if the .H file changed; see also **-c**
-B	Builds all targets regardless of file dates
-c	Caches autodependency information, which can improve the speed of MAKE. Use with **-a**; don't use if MAKE changes include files (such as using TOUCH from a makefile or creating header or include files during the MAKE process).
-D*macro*	Defines *macro* as a single character, causing an expression **!ifdef** *macro* written in the makefile to return true
[-D]*macro*=[*string*]	Defines *macro* as *string*. If *string* contains any spaces or tabs, enclose *string* in quotation marks. The **-D** is optional
-d*directory*	Used with **-S** to specify the drive and directory MAKE uses when it swaps out of memory. The option is ineffective when used with the MAKER
-e	Ignores a macro if its name is the same as an environment variable (MAKE uses the environment variable instead of the macro)
-f*filename*	Uses *filename* or *filename*.MAK instead of MAKEFILE (space after **-f** is optional)
-h or -?	Displays MAKE options and shows defaults with a trailing plus sign
-I*directory*	Searches for include files in the current directory first, then in *directory*
-i	Ignores the exit status of all programs run from MAKE and continues the build process
-K	Keeps temporary files that MAKE creates (MAKE usually deletes them)
-m	Displays the date and time stamp of each file as it is processed by MAKE
-N	Increases the compatibility of MAKE with NMAKE
-n	Prints MAKE commands but doesn't actually perform them (helpful when debugging makefiles)
-p	Displays all macro definitions and implicit rules before executing the makefile
-q	Returns 0 if the target is up-to-date and nonzero if it's not (for use with batch files)
-r	Ignores the rules and macros defined in BUILTINS.MAK
-S	Swaps MAKE out of memory while commands are executed, reducing memory overhead and allowing compilation of large modules. This option has no effect on MAKER.
-s	Silent; suppresses onscreen command display before executing the command
-U*macro*	Undefines previous definitions of *macro*
-W	Writes the current specified nonstring options to MAKE.EXE making them defaults

Library and startup files

The following table describes the 32-bit run-time libraries and start-up .OBJ files provided by Borland; use these with TLINK32.

Library and startup files for 32-bit Windows applications

Libraries and .OBJs	Description
CW32.LIB	Run-time library for 32-bit Windows executibles
CW32I.LIB	Import library for 32-bit Windows run-time library DLL
IMPORT32.LIB	Import library for 32-bit Windows executibles
C0X32.OBJ	Startup code for 32-bit console applications
C0W32.OBJ	Startup code for 32-bit Windows .EXE modules
C0D32.OBJ	Startup code for 32-bit Windows .DLL modules
CW32MT.LIB	Run-time library for 32-bit multi-threaded Windows executibles
CW32MTI.LIB	Import library for 32-bit multi-threaded Windows run-time library DLL

Library and startup files

The following tables show the different .OBJ and .LIB files you need to use when linking 16-bit .EXEs and .DLLs. In addition to the files listed, you'll also need to link:

- IMPORT.LIB for all applications that use the Windows API.
- FP87.LIB or EMU.LIB for DOS applications that use floating-point math.

Library and startup files for DOS applications

Model	Regular startup module	Compatibility startup module	Math library	Run-time library
Tiny	C0T.OBJ	C0FT.OBJ	MATHS.LIB	CS.LIB
Small	C0S.OBJ	C0FS.OBJ	MATHS.LIB	CS.LIB
Compact	C0C.OBJ	C0FC.OBJ	MATHC.LIB	CC.LIB
Medium	C0M.OBJ	C0FM.OBJ	MATHM.OBJ	CM.LIB
Large	C0L.OBJ	C0FL.OBJ	MATHL.LIB	CL.LIB
Huge	C0H.OBJ	C0FH.OBJ	MATHH.LIB	CH.LIB

Library and startup files for statically linked 16-bit Windows .EXEs

Model	Startup for .EXE	Math library	Run-time library
Small	C0WS.OBJ	MATHWS.LIB	CWS.LIB
Compact	C0WC.OBJ	MATHWC.LIB	CWC.LIB
Medium	C0WM.OBJ	MATHWM.LIB	CWM.LIB
Large	C0WL.OBJ	MATHWL.LIB	CWL.LIB

Library and startup files for statically linked 16-bit Windows .DLLs

Model	Startup for .DLL	Math library	Run-time library
Small	C0DS.OBJ	MATHWS.LIB	CWC.LIB
Compact	C0DC.OBJ	MATHWC.LIB	CWC.LIB
Medium	C0DM.OBJ	MATHWM.LIB	CWL.LIB
Large	C0DL.OBJ	MATHWL.LIB	CWL.LIB

Library and startup files for dynamically linked 16-bit Windows .EXEs and .DLLs

Model	Startup for .EXE or .DLL	Run-time library
Large	C0DL.OBJ	CRTLDLL.LIB

A
ANSI Standard 23
AppExpert 5, 9
application development 9
    Windows 4

B
Borland Component Libraries (BCL) 12
Borland Database Engine (BDE) 19
BRIEF 30

C
C++ language 23
Class Library 13
ClassExpert 5, 10
Clipboard classes 15
CodeGuard 21, 29
command line 22
command-line options 32
compiler options 32
compilers 22
container class 18
cScript 8

D
database application development 19
debuggers 20
debugging watch window 8
Development Suite 29
diagnostics 14
dialog boxes 11
Doc/View 14
docking classes 15
documentation 27
DOS applications 25

E
EasyWin 25
editor 5, 6
event handlers 14
example programs 27
exceptions 14
Experts 5, 9

F
ForeHelp 30

G
graphics 11

H
Help 27

I
IDE debugger 20
InstallShield Express 29
integrated debugger 20
Integrated Development Environment (IDE) 4, 7, 8
internationalization 26
Internet programming 14

L
language extensions 24
libraries 7, 12, 48
linkers 22
localization 26

M
MAKE options 48
MAKE utility 22
menus 11
Microsoft Foundation Classes (MFC) 12

N
namespace support 23

O
ObjectComponents library 13
ObjectScripting 4, 8
ObjectWindows 5, 10
    library 13
OLE classes 15
online Help 27
OS/2 30

P
platforms 25
portability 25
precompiled headers 5
productivity tools 5
Project Manager 6
project options 7
PVCS version control 29

Q
quick reference 31

R
resource
    compilers 22
    linkers 22

shells 22
Resource Workshop 5, 11
run-time libraries 12
run-time type identification (RTTI) 23

S

scripting language 8
Services library 13
shell classes 15
Source Pools 6
SQL Links 19
Standard C++ Library 12
Standard Template Library (STL) 12
Style Sheets 7

T

TargetExpert 5, 7, 26
TCP/IP protocol 14
technical support 28
templates 23
TLINK32 options 35
Turbo Assembler 30
Turbo Debugger 20
Turbo Profiler 21

U

user interface
  components 11
  design 5

V

VBX controls 14, 29
videos 28
Visual Database Tools 19

W

Windows
  application development 4, 9
  Help development 30
Windows Sockets 14
WinSight 21
WinSpector 21
WinSys library 13

Borland may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1987, 1996 Borland International. All rights reserved. All Borland products are trademarks or registered trademarks of Borland International, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Printed in Ireland

BCP1350WW21770 1E0R0196
9697989900-9 8 7 6 5 4 3 2 1
C1

Borland

Copyright © 1995 Borland International, Inc. All rights reserved. All Borland product names are trademarks of Borland International, Inc. Corporate Headquarters: 100 Borland Way, Scotts Valley, CA 95066-3249, (408) 431-1000. Internet: http://www.borland.com CompuServe: GO BORLAND. Offices in: Australia, Canada, France, Germany, Hong Kong, Japan, Latin America, Mexico, The Netherlands, Taiwan, and United Kingdom • Part # BCP1350WW21770 • BOR 8855